1983

Signal processor interface simulation of the AN/SPY-1A radar controller

Kersh, Todd B.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SIGNAL PROCESSOR INTERFACE SIMULATION
OF THE AN/SPY-1A RADAR CONTROLLER

by

Todd B. Kersh

June 1983

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Signal Processor Interface Simulation of the AN/SPY-1A Radar Controller	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, June, 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Todd B. Kersh	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: June, 1983
	13. NUMBER OF PAGES: 94
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

database, simulation, multimicroprocessor, real-time, AEGIS, SPY-1A, Phased Array Radar, Ada, Program Development Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis reports on the design and implementation of a simulation of the Signal Processor Interface to the AN/SPY-1A Phased Array Radar Controller. Inherent to the simulation is the development of a representative time sensitive database of the targeting environment. The programming language Ada was utilized as a program development language in the design for the database. The developed Target Database utilizes the 20 mega-byte REMEX (Cont)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued) Block # 20

Data Warehouse 3200 memory storage unit. The simulation of the Signal Processor Interface will allow real time testing of the Naval Postgraduate School's AN/SPY-1A Radar Controller System Model.

Signal Processor Interface Simulation of the AN/SPY-1A Radar Controller

by

Todd B. Kersh
Captain, United States Army
B.S., United States Military Academy, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

ABSTRACT

This thesis reports on the design and implementation of a simulation of the Signal Processor Interface to the AN/SPY-1A Phased Array Radar Controller. Inherent to the simulation is the development of a representative time sensitive database of the targeting environment. The programming language Ada was utilized as a program development language in the design for the database. The developed Target Database utilizes the 20 mega-byte REMEX Data Warehouse 3200 memory storage unit. The simulation of the Signal Processor Interface will allow real time testing of the Naval Postgraduate School's AN/SPY-1A Radar Controller System Model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The AEGIS System is the Navys multi-faceted shipbcard weapcn ccntrol, decision making, and surveillance system. The engineering model began testing on the "Nortcn Scund" in 1977 and the AEGIS System joined the Fleet cn board the "Ticonderoga" in 1982. To date, the AEGIS System represents the newest fielded technology in the Fleet and possibly in the world. Since every design effort must at some time in the design determine what the target hardware will be fcr the system, the result is that all "new systems" dc not in fact utilize the most current electronic advances. In addition, the further design, testing, and linking of the many separately developed and tested modules further increases this unaviodable hardware gap. In the case of the AEGIS System, this is particularly true since we have seen a technological revolution cccur during it's development. The Large Scale Integrated Circuits (LSI), and now the Very Large Scale Ingrated Circuits (VLSI) are common in our off-the-shelf technology. The Naval Postgraduate School AEGIS Modeling Group has been investigating the use of new off-the-shelf VLSI technology that could provide significant savings in mcney and space while still fulfilling the system requirements of the AEGIS system. The AEGIS Modeling Group decided early in their study and emulative modeling to choose the AN/SPY-1A Phased Array Radar Controller as a modeling subset of the AEGIS system. The SPY-1A Radar represents a sufficiently difficult and real time sensitive module of the AEGIS system such that if it

can be successfully emulated, then it should be possible to similarly build the other modules comprising the total AEGIS system.

The AN/SPY-1A is a complicated and extensive system in its' own right. The two primary modules of the SPY-1A Radar Controller are the Radar Scheduler and the Track Processor. Previous thesis work has been done to model those two modules by Grant [Ref. 1] and Cech [Ref. 2] respectively. In addition, the two systems that depend on the AN/SPY-1A for data - the Weapon Control System (WCS) and the Command and Decision System (CD) - have been simulated by Boone [Ref. 3] in his thesis. The Signal Processor module is another module to be simulated such that the NPS SPY-1A Model subset can be fully interfaced and tested for real time capability and logical functioning. The initial design, development, and target environment simulation of the SPY-1A Signal Processor Interface is the intent of this thesis.

B. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempting to cite each individual occurrance of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel Corporation, Santa Clara, California:
Intel, Intel 8086, iSBC 86/12A, MULTIBUS

Digital Research Corporation, Pacific Grove, California:
CP/M, CP/M-86, PL/I-86, PL/I-80, ED, RASM-86, LINK86, DDT-86

EX_CELL_C Corporation, Irvine, California:

REMEX Data Warehouse

MicroPro International, San Rafael, California:
Wordstar

Department of Defense, Washington D.C.:
Ada

Micropolis Corporation, Chatsworth, California:
Micropolis

Lear Siegler, Inc., Anahiem, California:
ADM-3A

C. PURPOSE OF THIS THESIS

The broad direction of the Signal Processor simulation is threefold:

1. Emulate the SPY_1A Signal Processor using the Remex Data Warehouse (a 20 megabyte fixed Winchester technology disk system).
2. Be able to emulate the signal processor functions to provide a real time test environment for the SPY-1A Model.
3. Be able to use the simulation to test the logic of the NPS SPY-1A Model.

These broad objectives were further subdivided into tasks to develop a target database module and two system testing modules. The first system testing module will emulate the hostile environment of targets utilizing a pre-developed target database while the NPS SPY-1A Model is being run/tested for real time operations. This model is designed to respond as quickly as possible to a dwell command from the Radar Scheduler Module with an appropriate data output to the Track Processor Module, to allow an accurate test of the speed of the overall SPY-1A Model. This design will be

herein referred to as the "Static Model". The second system will be used to test the logic of the internal SPY-1A modules, and will not be constrained to real time run requirements. It will display a target environment as it developes and changes over time, and allows the user to initiate and change the environment as he desires. This design will be herein referred to as the "Dynamic Model". The tasks for each of the Modules are as follows:

TARGET DATABASE:

1. Create targets.
2. Develop target tracks and record those target locations on the respective track at descrete time intervals in the database.
3. Modify and Delete targets and target dadta on the database.

STATIC Model:

1. Interface database access with NPS SPY_1A Model
2. Monitor the I/O interface during testing without detracting from the real time environment

DYNAMIC Model:

1. Allow interactive changes to be made to the database during runtime
2. display the tracks in the database as the simulation runs

The scope of this thesis extends primarily to the Target Database and Static Model development and implimentation, although the overall design structure is such that the Dynamic Model can encompass and utilize the modules developed herein.

D. THESIS ORGANIZATION

The thesis is organized into four chapters. The computer code developed to implement the system is contained in the following appendices. The first chapter covers the background of the Aegis Project at the Naval Postgraduate School, the basic direction for this thesis, and thesis organization. The second chapter covers the design of the Signal Processor Interface Module. It will discuss the overall considerations for the design, the interfaces neccessary between the Signal Processor and the other modules previously developed, and the specific design for the Target Database and Static Model. The programming language Ada was utilized as a Program Design language (PDL) in the development of a design for the Target Database and a Dynamic Model. The third chapter will discuss the implementation of the design for the Signal Processor Interface Module. Translation considerations when Ada is used as a PDL and the implementation language is PL/I are highlighted. The modules that make up the Target Database and the Static model are discussed in detail. Finally, Chapter four presents some conclusions on the work involved in the design and implementation of the Signal Processor Interface Module as it is now, how it might be utilized and changed by future Aegis Group members, and what the next logical steps should be toward the complete simulation of the critical paths of the SPY-1A Phased Array Radar Controller.

II. DESIGN OF THE SIGNAL PROCESSOR SIMULATION

A. OVERALL CONSIDERATIONS

1. Designing for Change

Tc provide the desired future maintainability and flexibility as a simulative and emulative instrument, it is neccesary to design the Radar Signal Processor Simulation with the capability for change. The latest concepts of good software engineering principles explain that forseeable and non-forseeable changes are sure to be applied to any software engineering project, but especially in those cases were the program being developed is being separately designed and implemented to become part of a larger system. To provide that capability, the designer and programmer must from the start try to separate those items that are likely to be changed and use the concepts of clear documentation and structured programming to make it easier for the system users and maintainers to incorporate changes. The decisions that are made in modularity and implementation must be documented to enhance the understandability of the system. As much as possible, the assignment of parameters and constants should be clustered or at least positionally standardized within modules to allow ease in finding them and changing them. The design concept of information hiding needs to be utilized such that enhanced versions of specific implementations can be easily substituted without causing major changes throughout the other modules that constitute the overall design.

2. Designing for Extensibility

A part of designing for change is the consideration of and provision for extensions to the basic design one may provide. It is important to consider the critical items in the design, and yet still allow for the addition of other modules that may provide desired functions for future users of the system. One way to provide this capability in a design is in utilization of a tree like structure that will allow the addition of other branches at any node of the hierarchy. In so doing, the designer offers the maximum flexibility in the basic design, and enhances future maintainability and changability, while providing for the unforseeable.

3. Modular Design

Incorporating the design principle of modularity will provide the basis for both changability and extensibility. Choosing modules in the program that describe a concise function, and interface with each other without side effects will enhance the understandability of the design and the resultant code. Utilizing a design methodology that incorporates the principles of top-down design and assists in the partitioning of a complex problem into intellectually addressable sub-problems will naturally produce good modularity. Boochs' "Object-Oriented Design" methodology [Ref. 4] discussed in detail in Section II.C and Appendix E provides these attributes. The design of both the target database and static model incorporates many of these principles limited only by the author's designing and programming prowess.

B. INTERFACES

In the thesis work done by Riche and Williams [Ref. 5], the overall design and modular interfaces were discribed and defined for all future SPY-1A Controller Module development. However, since their design incorporates the development of all the modules, and the project at this stage of implementation has only developed the most critical modules required to emulate a basic subset of the real SPY-1A Radar Controller, the interfaces they developed are not completely appropriate. To interface with the dwell commands passed by the Radar Scheduling module [Ref. 1] and to pass feedback that can be understood by the Track Processor module [Ref. 2], the Signal Processor Module must logically incorporate the Radar Output, Radar Return, and Beam Stabilization modules. Therefore, the interface utilized for input is table_58 ("Common Memory Interface between the Radar Scheduling and the Beam Stabilization modules"), and the interface used to output is table_8 ("Common Memory Interface between the Beam Stabilization and the Track Processor modules") [Ref. 5]. Not only will this extension of the logical interface for the Signal Processor make the future work of interfacing the modules easier, but it makes the present design for the Signal Processor Interace Module easier to implement. The chosen interfaces will allow the signal processor model to receive and send target data in terms of cartesian coordinates rather than the lengthy and complicated codes that specifically tell the signal processor where to point its beams. Tables I and II show the respective interfaces.

TABLE I

Signal Processor Output Interface

```
/*
OWNER: AEGIS MODELLING GROUP
DATE OF LAST UPDATE: 28 OCT 81
MODULE TYPE: TABLE
PURPOSE: COMMON MEMORY INTERFACE
NAME: E_TO_P_TABL
*/

/*
THIS TABLE INTERFACES BETWEEN THE BEAM STABILIZATION
PROCESS AND THE TRACK PROCESS
*/

declare
  1 B_to_P_tabl  static external,
    2 x_sub_s                    fixed bin(15)  initial(0),
    2 y_sub_s                    fixed bin(15)  initial(0),
    2 z_sub_s                    fixed bin(15)  initial(0),
    2 face_id                    fixed bin(15)  initial(0),
    2 dwl_idx                    fixed bin(7)   initial(0),
    2 trk_num                    fixed bin(7)   initial(0);

/* END OF TABLE */
```

C. USE OF ADA AS A PROGRAM DESIGN LANGUAGE

Grady Booch [Ref. 4] has proposed a software engineering design technique he terms "Object-Oriented Design". Although his chosen name for the design methodology may be unfortunate considering the controversy raised by the ambiguous term "Object", and the past use of the term in reference to the Smalltalk programming language, the design methodology itself works well. Using the new Department of Defense programming language Ada, the purpose of Object-Oriented Design is to produce logical, efficient, highly readable and understandable code that accurately reproduces the real world problem in the computer space. Ada is utilized as a program development language because of

TABLE II

Signal Processor Input Interface

```
/*
OWNER: AEGIS MODELLING GROUP
DATE OF LAST UPDATE: 2 NOV 81
MODULE TYPE: TABLE
PURPOSE: COMMON MEMORY INTERFACE
NAME: R_TO_B_TABL
*/


/*
THIS TABLE IS THE INTERFACE BETWEEN RADAR SCHEDULING
AND BEAM STABILIZATION PROCESSES
*/


declare
  1 R_to_B_tabl (10)          static external,
    2 search_dwls,
        3 asim                          fixed bin(15)  initial(0),
        3 elev                          fixed bin(15)  initial(0),
        3 time
          4 msb                         fixed bin(15)  initial(0),
          4 lsb                         fixed bin(15)  initial(0),
        3 dwl_idx                       fixed bin(7)   initial(0),
        3 beam_purpose                  fixed bin(7)   initial(0),
        3 alpha_delta_cos_offset        fixed bin(15)  initial(0),
        3 beta_delta_cos_offset         fixed bin(15)  initial(0),
        3 face_assign                   fixed bin(7)   initial(0),
    2 trk_burnthru,
        3 type_dwl                      fixed bin(7)   initial(0),
        3 stabe_coords,
          4 x_stbl                      fixed bin(15)  initial(0),
          4 y_stbl                      fixed bin(15)  initial(0),
          4 z_stbl                      fixed bin(15)  initial(0),
        3 dxl_idx                       fixed bin(7)   initial(0),
        3 time,
          4 msb                         fixed bin(15)  initial(0),
          4 lsb                         fixed bin(15)  initial(0),
        3 beam_purpose                  fixed bin(7)   initial(0),
        3 alpha_delta_cos_offset        fixed bin(15)  initial(0),
        3 beta_delta_cos_offset         fixed bin(15)  initial(0),
        3 face_assign                   fixed bin(7)   initial(0);

/* END OF TABLE */
```

its capabilities in the production of highly structured and modularized algorithms. It also has the nice feature of separating the specifications for the modules utilized in the design from the actual methods used for implementation

cf thcse mcdules, and thus provides the designer with an ability to postpone the implementation decisions for as long a time as convenient during the design phase. This feature was particularly important since the programming language PL/I-86 was going to be used for actual implementation, and it was intuitively felt that some design changes and concessions might have to be made even at the highest levels because of the differences in the large scale data structures provided by the two languages.

Object-Oriented Design methodology is broken down into three basic steps:

1. Define the Prcblem
2. Develop an Informal Strategy
3. Fcrmalize the Strategy

"Defining the problem" involves the development of a concise paragraph in English that specifically outlines the real world problem. "Developing an Informal Strategy" is to develcp an English paragraph that as clearly and concisely as possible describes how one will solve the problem. This second step is really the most difficult part of the methodology, since the resultant success of the design rests on how well this can be accomplished by the designer. The last portion "Fcrmalize the Strategy" is where the algorithm begins to take shape. First, the designer must pick out the proper ncuns that describe the "objects" of the solution strategy. Those objects are discribed in terms of major objects and attribute objects. Next, the Informal Strategy is again scrutinized, this time to pick out the verbs that represent the "operations" utilized in the solution strategy. These operations are then grouped with the objects they logically affect in the informal strategy. Bcoch then would have the designer draw an object-oriented system graph depicting the objects as Ada "packages" and the operations as Ada procedures and functions within the packages. The

object-oriented system graph describes the hierarchial interfaces between the structures. Booch typically includes an Ada "subprogram" as a controlling program that utilizes the develcped packages. Finally, the package specificaticns are written in Ada utilizing the previously developed cbject-oriented system graph as a guideline for the interfacing and specific procedure specification development. This was done in the design of a "Dynamic" model and Target Database system and is specifically shown in Appendix E for future use by the AEGIS Modeling Group.

D. THE DATABASE

Using the Ada design as a basis, the Target Database was designed for implementation in the PL/I and ASM-86 languages.

1. Interfacing and Storage

The AEGIS Mcdeling Group experimental computer depends cn a 32 k byte "common memory" board on the MULTIBUS to pass messages. The common memory board is further utilized for the commands to the REMEX Data Warehouse 20 mega-byte storage system and the buffered data items to be written cn and retrieved from the REMEX. A mapping of how the ccmmcn memory is currently partitioned is shown in Figure 2.1 The segmented memory base for the common memory is 0E000 hex and offsets are as shown. The REMEX Data Warehouse is also connected to the MULTIBUS and data is transfered to and from it in response to the formated messages. The processes for the operations of the REMEX are discussed in detail in the thesis work done by Almquist and Stevens [Ref. 6] and in the appropriate manuals [Ref. 7]. The basic message fcrmat utilized for this thesis is the read/write format [Ref. 8] (as specified in Figure 2.2).

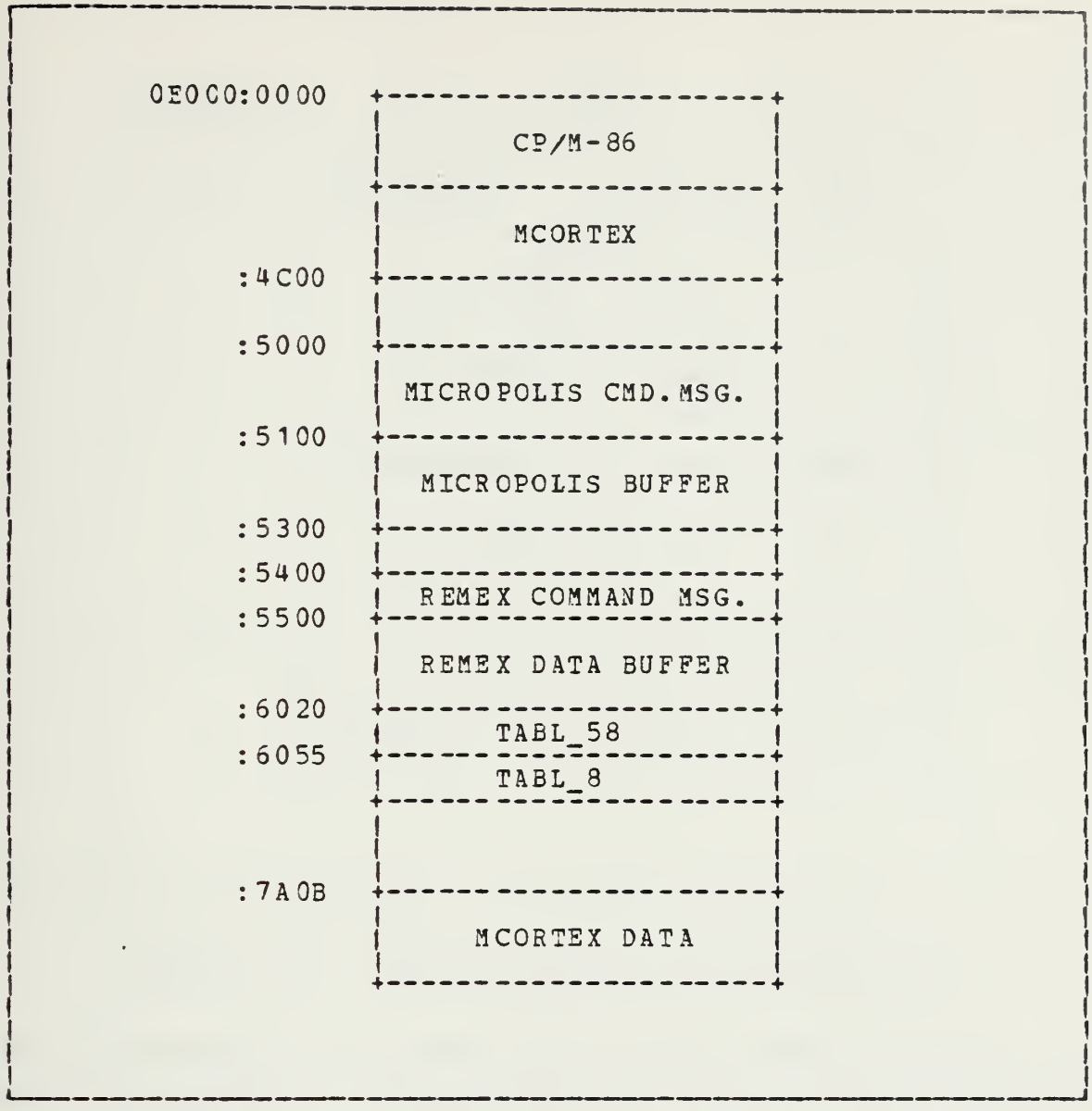

Figure 2.1 Common Memory Map.

2. Design Decisions

The Ada design for the Target-Database resulted in a system that protects and hides the actual database and how it was implemented from the user. In an effort to incorporate that design feature in the PL/I-86 implementation, the concept of using a specific module to perform all target

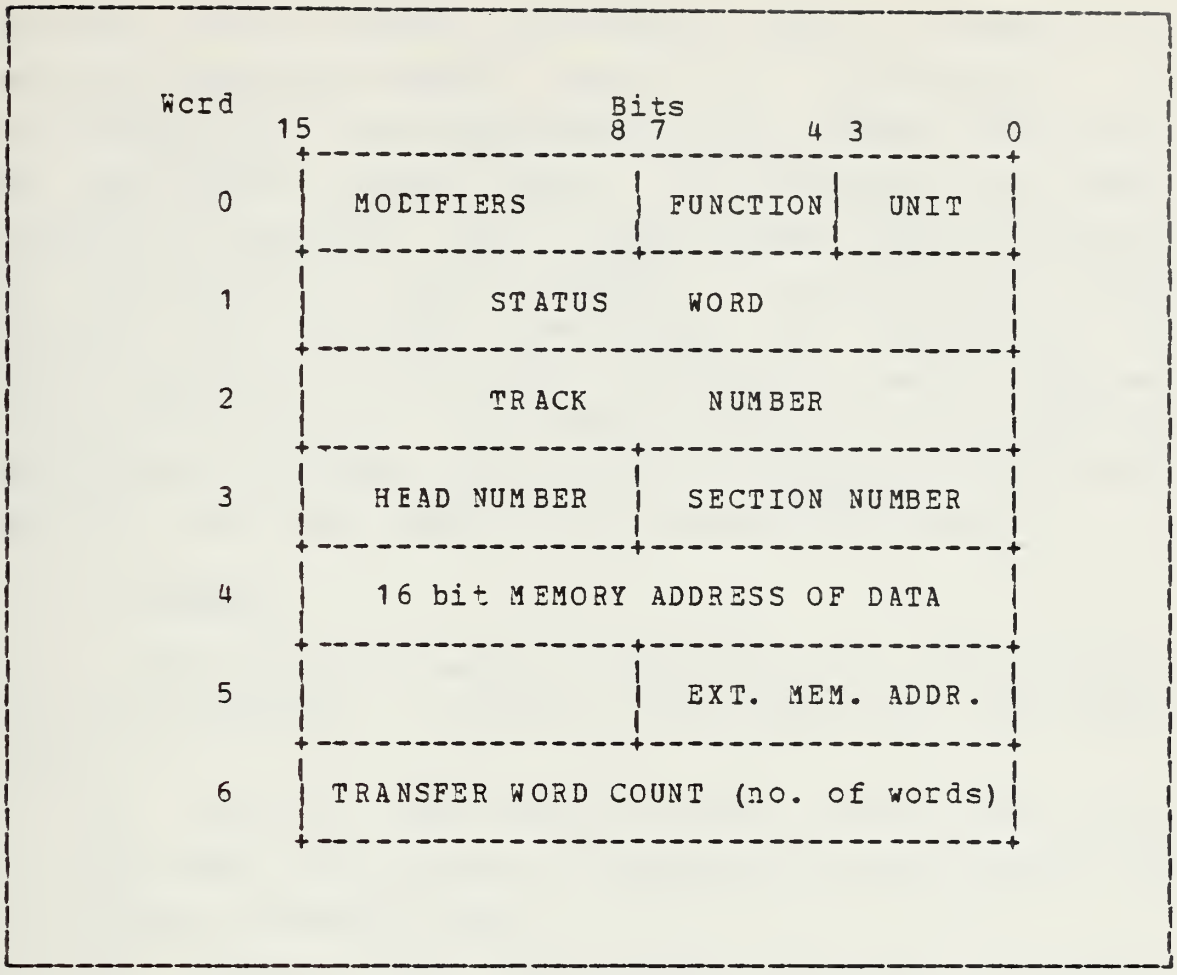

Figure 2.2 REMEX Read/Write Message Format.

data conversion to an appropriate output message format, and then to perform the operations to place that formated message in the REMEX was conceived. This module is named Bld_database. Not only does it perform those tasks just mentioned, but because of it's modularity, it provides for future changes should the method of storing the database be modified, or the hardware device utilized for storage be replaced. In addition, the decision was made to store only the messages to be utilized for output on the REMEX, rather than storing both the initial target data that produced the messages and the messages themselves. The reason for this

decision is to provide the fewest number of REMEX data seeking operations during the run of a simulation. By utilization of a data structure on the current iSBC 86/12A in RAM (Random Access Memory) to pre-build the proper sequence of messages, only a write command will be required to retrieve data from the REMEX. Although this method requires more execution time during the creation of the Target-Database, very little execution time is consumed during the emulation. The method utilized for creation and modification requires the partial re-building of the database for each change made to the Target-List of data.

3. Design

The resultant design, modularized for implementation in PL/I procedures, consists of the following (see Figure 2.3):

a. Control: This module contains the main menu where the user will be able to choose how he will utilize the Signal Processor Model.

b. Create: This module allows the user to interactively construct the initial environment of targets and how they will change throughout the time of the simulation.

c. Delete: This module allows the user to delete targets from the environment at any desired simulation time point.

d. Change: This module allows the user to change the environment during the initial creation of the environment.

e. Build_Database: This module is not called by the control module, but interfaces between the database utilized to represent the environment and those modules utilized through "control" to initially create and further modify and run the simulation. It must be able

to build a set of output messages based on the previously developed target data, and to send those messages to the REMEX for storage.

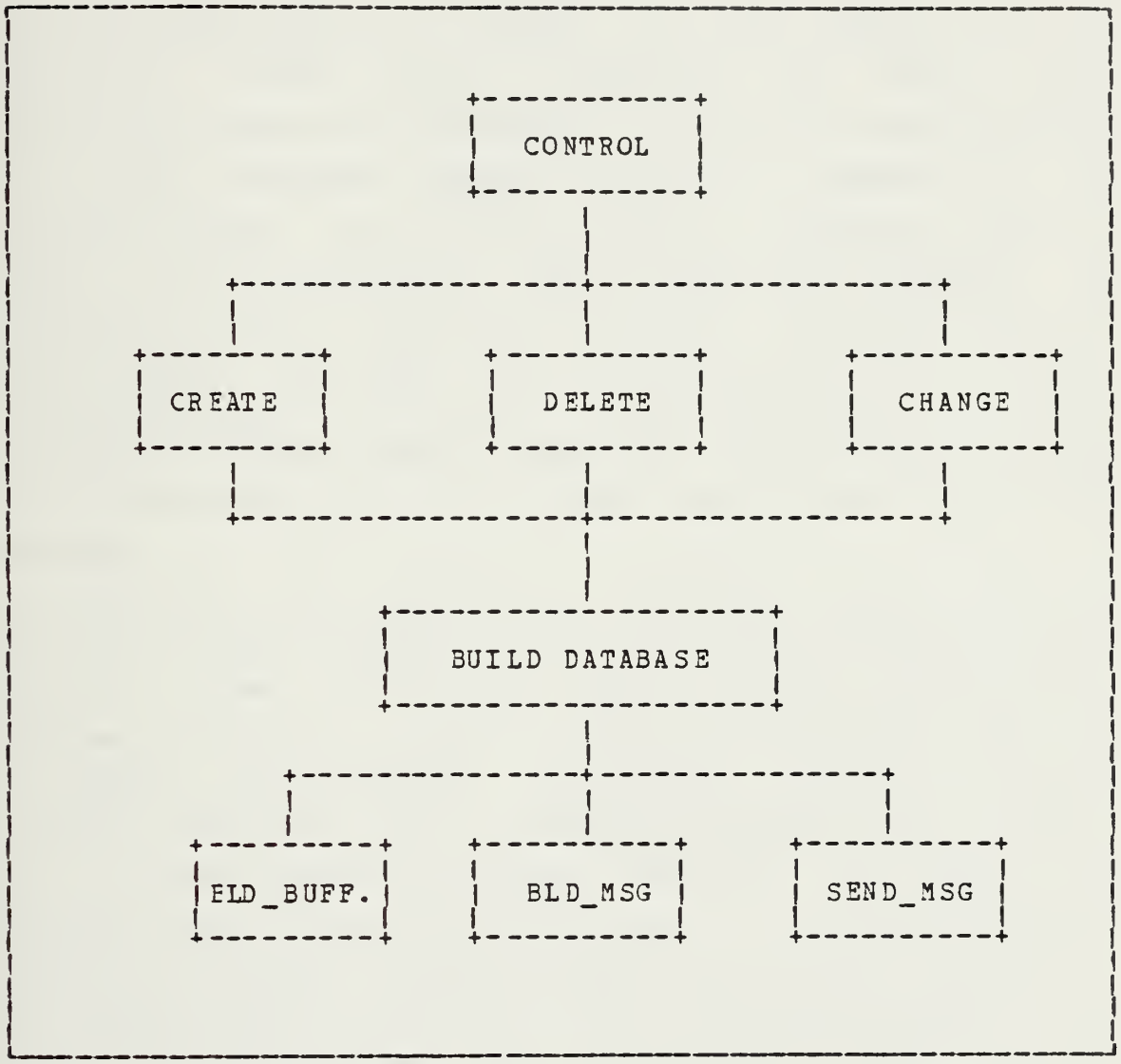

Figure 2.3 Database Design.

E. THE STATIC MODEL

The design of the Static model depends on the ability to read sequentially arranged previously stored output messages from the hard disk. When keyed by a dwell command from the SPY-1A Track Scheduling module, an output message will be placed in common memory providing the SPY-1A Radar Controller system with feedback. It does not matter whether the output message offers a "logical" response to the requested dwell, just that it offers a response that will cause the SPY-1A System to send another dwell command. By timing the SPY-1A System as it runs in interface with the Static Model, the user will be able to acertain the real-time performance of the SPY-1A model and whether or not the concurrent multiproccessor system can indeed operate within the specifications of the AEGIS SPY-1A Radar Controller system.

Utilizing the previously discussed target database design, a Target Database to be utilized by the Static Model can be created. The Static Model consists of functional modules that must be able to retrieve sequential data from the REMEX Data Warehouse, and respond to each new dwell command (tabl_58) sent by the Radar Scheduler with a set of one or more feedback messages (tabl_8) that would have resulted from a simulated radar dwell. To enable the capability to time the turnaround speed of the SPY-1A Model, a CRT display will be required that allows measurements to be made. It is important that the display include only the minimum data so that it will not impede the performance of the Static Model, and thereby detract from the objective of measuring the SPY-1A System Model real-time performance. Figure 2.4 shows the Static Model functional modules in a hierarchial design. It is envisioned that the concurrent activity of the SPY-1A Radar Controller and the Signal

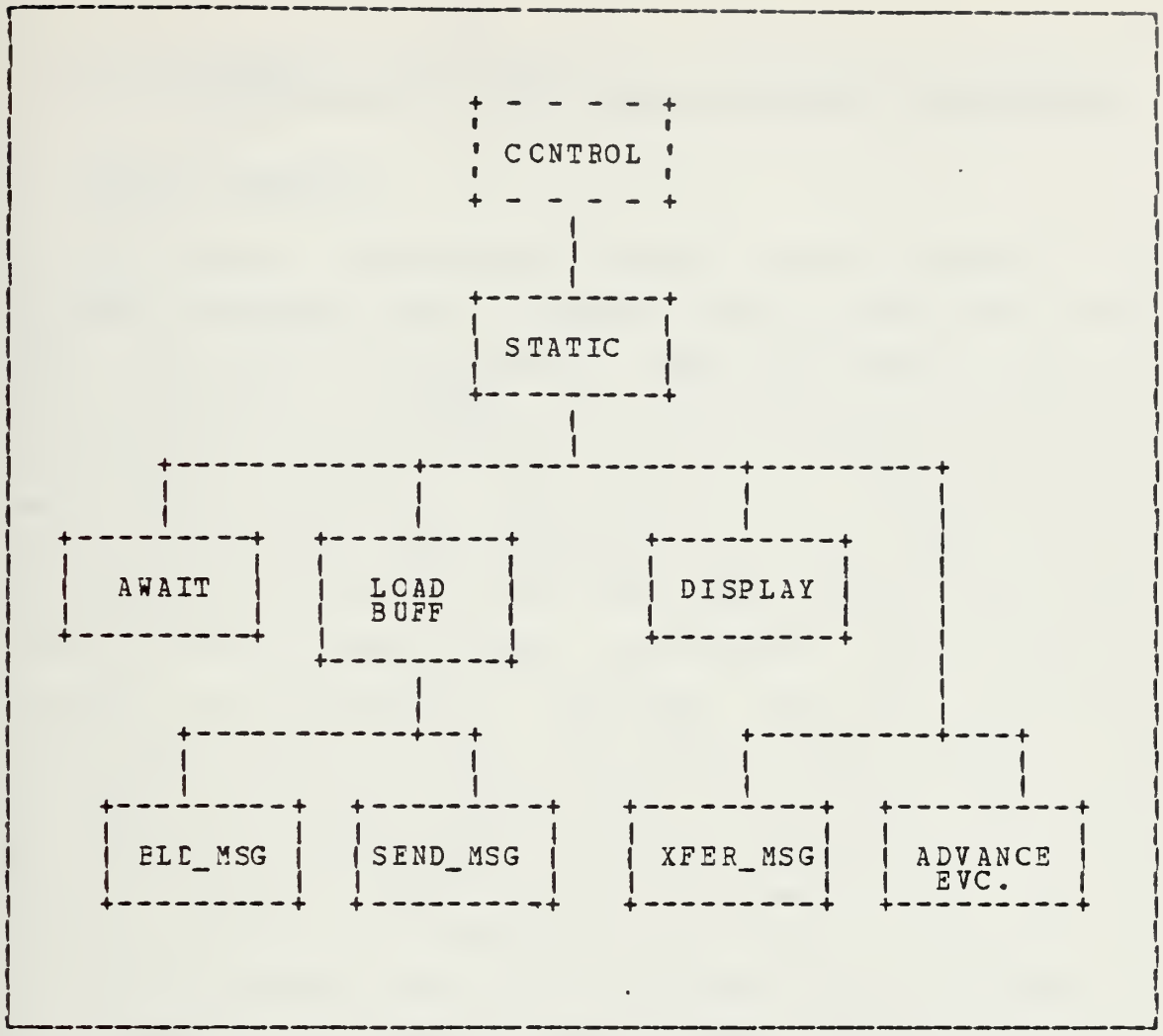

Figure 2.4 Static Model Design.

Processor Simulation will be sequenced using the MCORTEX Operating System functions [Ref. 9] implementing Eventcounts and Sequencers. Thus, although that interface is not available now, the AWAIT and ADVANCE primitives will be used at some future date by the AEGIS Modeling Group. In the meantime, in keeping with the design goal of changeability (separating and modularizing those items likely to be changed), the AWAIT and ADVANCE primitives must still be incorporated in the design and implemented for proper system testing.

III. IMPLEMENTATION OF THE SIGNAL PROCESSOR SIMULATION

A. TARGET HARDWARE

The present experimental computer system consists of a MULTIBUS backplane that contains enough space for twelve (12) Intel SBC 86/12's (Single Board Computers), four (4) ADM-3 terminals connected to the four (4) currently installed iSBC 86/12A boards and two different hard disk memory storage devices. (see Figure 3.1). The main storage device is the Remex Data Warehouse disk unit [Ref. 8] which contains two standard 8 inch IBM format floppy disk drives (one of which is used to boot the system), and a four head fourteen inch Winchester technology hard disk containing twenty mega-bytes of store. The other storage device is the Micropolis Hard Disk system [Ref. 10] which has five heads and contains an additional thirtyfive mega-bytes of storage space. In both storage systems, the user, under the CP/M operating system, is allowed to write only to the disk that the terminal device was initially logged into, although full read capability across all fixed storage devices is allowed. Shared memory consists of 32K bytes of Random Access Memory (RAM) that has been assigned the base address of 0E000:0000 hexadecimal. Occupying one of the twelve board slots, there is also a non-volative bubble memory which was in the past utilized for the boot precedure during initialization [Ref. 6] but is currently utilized as temporary storage to boot the operating system into each of the iSBC 86/12 boards in use.

The Intel SBC-86/12's use an 8 Mhz clock and contain 64k of internal memory that can be used for on board processing. Each of the iSBC 86/12's is connected to an ADM-3 terminal

that is used for communication. The operating system is Digital Research's CP/M-86 [Ref. 11] as modified by previous thesis students [Ref. 6] to enable the sharing of peripheral

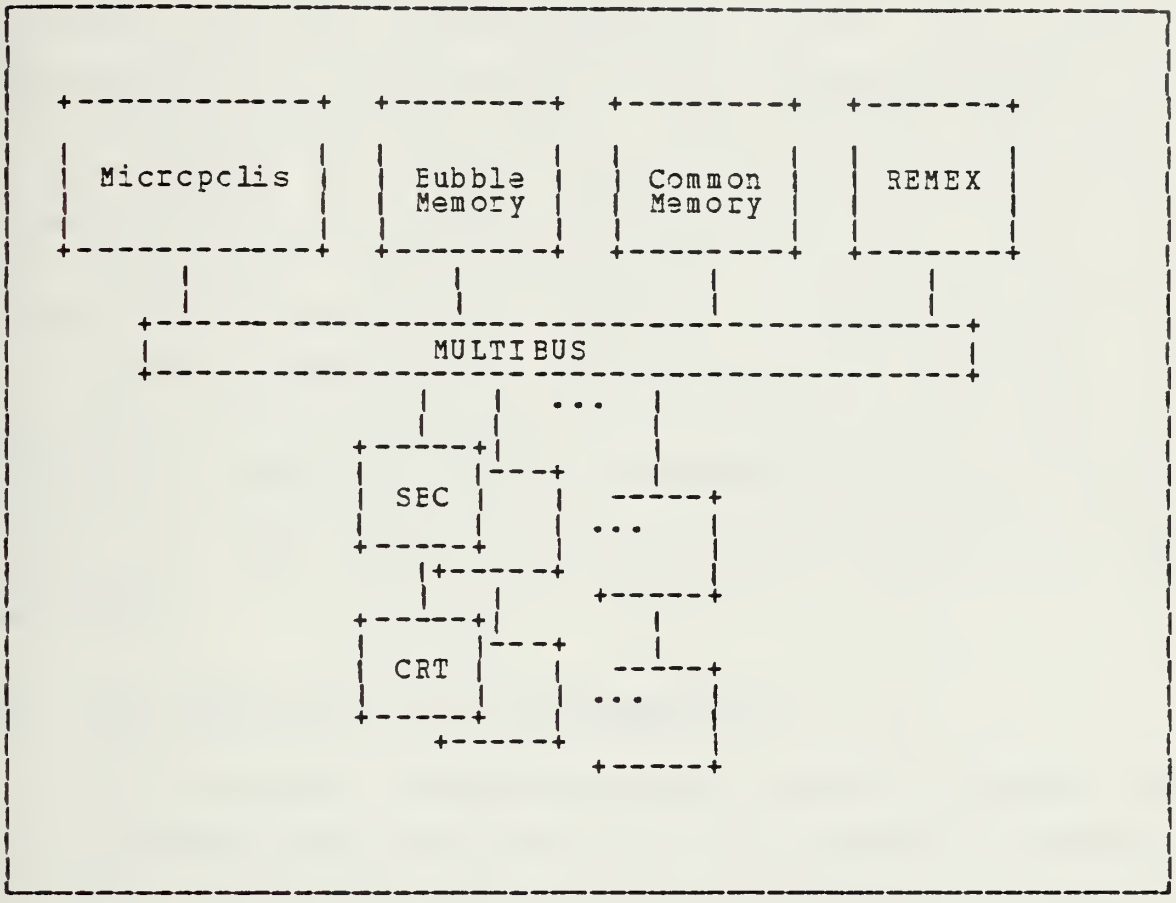

Figure 3.1 NPS AEGIS Modeling Group Experimental Computer.

devices. There is an executive called the "Multicomputer Real Time Executive" (MCORTEX) [Ref. 9] that has been written tc allow fcr concurrent computation by the SBC's. It occupies close to 6k bytes of storage on each of the SBC's. It is projected that it will take aproximately 8 or 9 SBC's to carry out the same processes that the four AN/UYK 7's presently do in the Spy-1A Radar Controller.

B. SOFTWARE DEVELOPMENT ENVIRONMENT

Recent aquisitions by the NPS AEGIS Modeling Group have made the Software Development Environment available on the experimental computer much better. The multicomputer system operates under the CP/M-86 operating system. In the past, programing has been done with Intel's PLM-86 Compiler and the ASM-86 Assembler for use on the iSBC 86/12. The SPY-1A major modules have been written utilizing the PL/I-80 Compiler based on the Intel 8080 microcomputer. Now, the PL/I-86 Compiler [Ref. 12] has been released and is available for programming use. Because of the requirement for 128 k-bytes of RAM for the use of the PL/I-86 Compiler, it can only be utilized by one of the four users at a time. In addition, where in the past programmers have gone to great lengths to avoid the use of the only available CP/M-86 text editor ED, the full screen text editor WORDSTAR is now available.

C. ADA DESIGN VS PL/I-86 IMPLEMENTATION

The previously discussed design process utilizing Ada was insightful and useful as a tool for program development, but the implementation language for the AEGIS Modeling group is PL/I. The primary structure resulting from the object-oriented design methodology is the Ada "package". The package in Ada serves to promote data abstraction and information hiding. PL/I does not offer a construct similar to Ada's "package" structure, but abstraction of the data manipulation and hiding the form of the implemented database can readily be achieved. The modules that are contained within the Ada packages are written as a logical grouping of procedures - the primary module structure in PL/I. The subprogram utilized as a control program in the Ada design is logically implemented with a controlling procedure, the

"procedure options (main)" in PL/I. The Ada package containing the target information is implemented with the global declaration "DBASE.DCL" and the resulting linked list used to develop the Target-List. The Ada database package is implemented with the PL/I array data-structure "BUFFER", which is hidden within the "BLD_DATABASE" procedure. The "BLD_DATABASE" procedure is not accessible directly to the user, and thus further hides the form of the database.

D. MODULES OF THE TARGET DATABASE

1. General Comments

A useful feature in PL/I is the "%INCLUDE" statement which allows one to make the compiler include programs or declarations that have been previously written. This feature is most commonly utilized to include declarations that are used by more than procedure throughout a system. The "global declarations"(DBASE.DCL) utilized in the Target-Database modules are treated in this manner. Future maintenance on the Signal Processor Interface Simulation that may modify the Target-List, can be made to DBASE.DCL, and after the program is re-compiled and re-linked, it will appropriately affect all the pertinate modules. In addition, two global variables within the functional grouping of the Target-Database modules are defined. These variables are declared as "external" initially in the main procedure "Control", and are also part of the global declarations utilized by the Target-Database. The two variables are "delta_t" and "endtime", and should be noted and protected appropriately by any future changes made to the modules of the Signal Processor Interface Simulation. It should be noted that "delta_t" is not utilized by either the Target-Database or the Static Model, but has been included because it will impact on the future use of the Signal

Processor Interface Simulation. It is meant to define the ratio of how many dwell commands are received versus the number of times the database buffer in common memory is updated. "Delta_t" is meaningful for the Dynamic Model and will impact on the length of time a test run will be able to run (based on available memory space for a database and chosen delta_t).

2. CONTROL.PLI

The Control procedure is the head node of the hierarchial structure of procedures used to modularize and structure the implementation of the Radar Signal Processor Interface Simulation. It contains the Main Menu that the user will be continually coming back to to route himself to other functional branches of the tree-like system. The Pl/I exception handler ON <condition> <body> is utilized first here and throughout the other modules to prevent abrupt program termination and promote graceful recovery in the event of user entry errors. Within the ON-body a series of IF-THEN statements are used. These will allow one to determine in which interactive block the error was committed. The variable named "block" is set to different integer values throughout the program to signal where the user is, and where is the appropriate place in the program to return the control, so that interaction can continue. The reader may be aghast at the flagrant and apparently unstructured use of "go to"s in this and further modules within the On-body exception handlers. One should be assured that exception handlers are probably the only generally acceptable and appropriate time to use the "go to" in a structured program. PL/I addditionally offers a further exception handling feature, the SIGNAL <condition> command. When used in conjuction with an IF <condition> THEN <statement> control command, the signal command has the

effect of "signalling" the system that the defined error (the condition of the signal call) has occured. The control is transfered automatically to the appropriate ON-unit defined for that signalled error. This enables the programmer not only to gracefully react to defined system errors, but to define his own error conditions and gracefuly continue operations. The Control module and the other modules in the Signal Processor Emulation utilize this feature to prevent the user from entering a response outside the defined allowable range. Finally, the REVERT <error type> statement is required at the end of each module where the ON exception handler is utilized. A stack is utilized in PL/I to save the state of the current ON-conditions when calling another procedure. PL/I-86 allows sixteen nested ON-units on the stack. Proper utilization of the REVERT command will pop the stack appropriately to ensure that the proper ON-unit is used. Functionally, the Control procedure sets up the global variable "endtime" that is utilized by the other Target-Database modules (create, delete, change, printlst, and bld_database) to define the limits of time for which the database is to be constructed on the REMEX Data Warehouse.

3. CREATE.PLI

The Create procedure has the function of interactively contructing a linked-list (the Target-List) of target nodes that contains the data for the database of discretely timed output messages (tabl-8). The linked list utilizes a pointer to the header node (appropriately called "head") that will be used by the other modules in their subsequent manipulations of the Target-List. The other two pointers utilized (tgt_ptr, and tgt_mkr) are used to traverse the linked list and manipulate fields on nodes, or nodes themselves. The PL/I "%INCLUDE" statement allows the

declaration of the linked list structure "Target" without having to declare it in every module it is referenced. The Target-List is implemented as a linked list to allow the most efficient use of storage during the run-time environment of the system. PL/I will only initiate storage for the nodes of the linked list during run-time, as required by the "ALLOCATE" command. Thus, instead of being restricted to a particular array size (had that data structure been used) as allocated at compile time, the user is restricted to the available memory at that time. In this version, the Target-List is constrained to 56 nodes (or targets), since the buffer size and corresponding sector size of the REMEX Data Warehouse is fixed at 512 bytes. After the user stops building the Target-List, the initial Target-Database is constructed with a call to the "Bld_database" procedure. Note that the first parameter to Bld_database is a constant "1". This will ensure that the first database built on the REMEX begins at the first discrete delta_t time value.

4. DELETE.PLI

The purpose of this module is to delete target nodes from the Target-List as requested by the user. The user has previously interactively indicated the specific discrete "time_in" value of the call, and this information will be further utilized by the procedure "Bld_database" in its call by Delete. Delete will request a target node number of the node to be deleted from the Target-List. Then, the pointers tgt_ptr and tgt_mkr are utilized to traverse the linked list until the appropriate target node has been located. When found, the target node is separated from the Target-List, and placed back in available free memory store by the use of the PL/I "FREE" command. If the target node can not be found (indicated by the pointer tgt_ptr reaching the "null" node), the user will receive an error statement and the

While loop controlling this process will return the user to ask if there is another node to be deleted. When the user has deleted all the targets he desires, the call will be made to the "Bld_database" procedure to re-build the database from the previously defined delta_t discrete time increment ("time_in") to the previously defined "endtime". Control will then return to the Main Menu (the Control procedure).

5. CHANGE.PLI

The Change procedure is similar to the Create procedure since it allows the user to re-define the fields of any given node on the linked list Target-List in an interactive mode. Once again, in a manner similar to the Delete procedure, the user will define the discrete delta_t time value where the Target-List is to be changed, prior to the call to Change. This value "time_in" will be passed to the "Bld_database" procedure in the same manner as with Delete for further Target-Database re-construction. The Change procedure allows the user to not only change the parameters used by the defined parametric equation but to change the equation (and therefor the shape of the resultant track) itself. The user is placed in a While loop to change all the targets he desires on the Target-List, until he indicates he is finished. At that time the procedure "Bld_database" is called to re-build the Target-Database on the REMEX Data Warehouse from the time given in the first parameter "time_in" to the "endtime", both previously determined by the user.

6. BLD_DATABASE.PLI

This module is the real workhorse of the Target-Database building system. The purpose of the Bld_database module is to convert the data contained on the

Target-List to an Array of output messages to be placed in a based structure called "Buffer", and then to transfer that Buffer to another buffer of equal size in the NPS experimental computer's common memory. At that time, the appropriate message will be sent to the REMEX Data Warehouse commanding it to read the data (using Direct Memory Access - DMA) onto the required track and sector of the REMEX hard disk. To accomplish these tasks, Bld_database utilizes three assembly language routines: Bld_buff, Build_cmd_mess, and Send_mess. Bld_buff utilizes the pointer to the structure "Buffer" and causes the structure to be copied into common memory starting at location 0E000:5500. Build_cmd_mess uses 8 parameters to build an appropriate REMEX command message formated for a "read" operation into common memory starting at location 0E000:5400. Send_mess tells the REMEX it has a command message at location 0E000:5400 and verifies that the REMEX has received and responded to the message. Bld_database utilizes these three primitive routines within two sub-procedures "bld_msg_buffer" and "call_rdw". The subprocedures themselves are called sequentially from within the execution of a PL/I DO loop that runs from the Bld_database parameter "time_in" to the global variable "endtime". The astute reader may now see why the user needs to build his Target-Database in a sequential manner, making deletions and changes in a progressively increasing discrete time increment up to "endtime". If modifications are not done in discrete sequential time, Bld_database will write over the changes that had been already written to the database with a higher value time increment than the current "time_in" defined. The sub_procedure bld_msg_buffer will utilize the Target-List to build a corresponding output (tabl_8) message to be incrementally placed in the Buffer structure sequentially as the linked list is traversed. Reaching the "null"

node will cause the while loop to end, and a call to the bld_buff primitive routine to be made. The x, y, and z named fields for each component of the Buffer array will be constructed by the parametric equation number indicated in the Target-List. These parametric equations were derived from a previous thesis work done by Boone [Ref. 3] and are utilized here to maintain overall SPY-1A system compatibility and integrity. See Figure 3.2 for a listing of the

```
(1)   x(t) = a + b*t + c*t*t
      y(t) = u + v*t + w*t*t
      z(t) = d

(2)   x(t) = a + b*t + c*t*t
      y(t) = u + v*sin(w*t)
      z(t) = d

(3)   x(t) = a + b*cos(c*t)
      y(t) = u + v*t + w*t*t
      z(t) = d

(4)   x(t) = a + b*cos(c*t)
      y(t) = u + v*sin(w*t)
      z(t) = d
```

Figure 3.2 Signal Processor Track Parametric Equations.

parametric equations. These parametric equations can easily be changed if desired by future users of this system, if specific requirements so dictate it. The sub-procedure load_rdw uses the primitives build_cmd_mess and snd_mess to cause the REMEX to read the data from the common memory buffer. Two of the parameters to the routine Build_cmd_mess require the track and sector to be designated where the REMEX will subsequently store the common memory buffer. To ensure that the track and sector are located in a sequential and therefore easily retrievable manner, a set of simple

algorithms were devised. The algorithms will require buffers to be stored starting at a location indicated by "time_in" and sequentially building each of 39 sectors per hard disk track until "endtime" is reached or the memory is depleted (at track 210). The algorithms are:

```
sect = 1 + mod(time_in,40)
track = 1 + trunc(time_in/39)
```

The "sect" algorithm will convert time_in to a modulo 40 number (0-39) and add 1 (since the sectors are number 1 to 39 per track). The "track" algorithm will divide time_in by 39 and truncate the resultant number to get an integer. It then adds 1 (since the REMEX does not allow the use of track 0 to the user). The subsequent calls are then made to build_cmd_mess and snd_mess in that order. Upon completion, Bld_database returns to the procedure from where it was called (Create, Delete, and Change) and then to Control to the Main Menu once again.

7. PRINTLST.PLI

The Print_lst procedure is meant to be a tool for the user to maintain a listing of the Target_List as the list is initially created and as changes are made during a database building session. The procedure will prompt the user to turn on the printer or hit <control> "P" to activate the printer, before typing "0" to begin a print of the Target_List. The Target-List print out will be initialized with a record of the time in ("time_in") for proper record keeping, and the linked list will be traversed, reading and printing the fields contained on each node. When the "null" node is reached, the procedure returns control to the Control procedure Main Menu.

E. MODULES OF THE STATIC MODEL

1. General Comments

The purpose of Static Model is to run through the developed Target-Database in as rapid a manner as possible, reponding to eventccunts from the SPY-1A Model (indicating a dwell command has been sent) by transfering a output message to common memory. The SPY-1A is then further notified that a message is ready fcr it's input by the advancing of a corresponding eventccunt. The display of the Static Model is merely a counter indicating each data transfer made (set of output messages) from the REMEX Target-Database tc common memory, and the anticipated endtime (or endpoint) for the Static Model simulation run.

2. STATIC.PLI

The Static procedure is the main procedure for the running of the Static Model. The procedure can operate in cne cf twc possible mcdes. The first mode is an actual run of the NPS SPY-1A Model as it will be eventually interfaced for testing. It is assumed that the MCORTEX cperating system will be utilized to enable proper interaction between concurrent processes, therefor eventcounts and sequencer primitives are used in the calls herein (which will be replaced by appropriate calls to that operating system at some future time). In the meantime, to allow testing of the Static Mcdel, an AWAIT primitive was written, and an ADVANCE primitive is utilized in the test program SPYTEST. The Static Mcdel will locp through the Target database in a PL/I DO locp from discrete time 1 to endtime. Within the loop, sequential calls are made to AWAIT, Load_buffer, Send_cutput, and Display. When the user begines a test-run with the SPY-1A Simulator, he will be prompted to load that program cn another iSBC 86/12 console, and then begin

operating. At that point, the same loop will be run as previously described. The user may also leave the Static Model and return to Control's Main Menu.

3. ICAD_BUFF.PLI

The purpose of this module is to extract the proper sector/track combination of data from the REMEX Target-Database, and place it in the common memory buffer. It is the same as the Bld_Buffer sub-procedure previously described as a part of Bld_database, except the parameters to the primitive routine Build_cmd_mess are to "write" instead of read.

4. XFER.A86

The purpose of this module is to transfer a output message (tabl_8) from the common memory buffer to common memory location starting at 0E000:6055.

5. ADV1EVC.A86

The purpose of this module is to advance an event-count in common memory to notify SPYTEST.PLI that the output message is ready to be read.

6. DISPLAY.PLI

The purpose of this procedure is to send to the terminal screen the "time" corresponding to the sequential transfer of sectors of data from the REMEX Data Warehouse, and show the user the expected endtime for that particular run. This should enable the user to determine the "real-time" capability of the SPY-1A Model.

F. ASSEMBLY, COMPILING, AND LINKING

The assembly language code was written in ASM-86 and assembled using RASM-86. This assembler produces relocatable files that can then be linked with compiled PL/I-86 files. The PL/I-86 Compiler was utilized for compilation of the PL/I programs, and the resulting assembler and compiler ".OBJ" files were then linked using LINK86. The LINK86 linker enables the user to develop a ".INP" file containing the list of program commands the user would normally have to type in, and the linker can then be optionally utilized with the command "LINK86 <file name>.INP [INPUT]". This greatly speeds the link process and assists during run-time testing and debugging. See [Ref. 12] for further information.

G. TESTING

Most of the implementation of the PL/I code was done using PL/I-80 instead of PL/I-86. This was convenient because of the extensive availability of microcomputers using PL/I-80 versus PL/I-86. Most of the early testing was done via extensive code reading and revision. As a result, during top-down testing of modules, (utilizing program stubs for the assembly language subroutines), the system worked with few runtime errors. Initially, the linked system did not contain the PRINTLST.PLI code it now incorporates. This code was developed as a test routine to insure that the Target-List and the Buffer data structures were being built in the proper manner and receiving the proper data. However the program was perceived as a desirable tool for recording target data while developing a Target-Database in the Signal Processor Interface Simulation, and was therefore incorporated into the system. The top-down testing philosophy enabled testing to be implemented in PL/I-80. This provided programming and testing flexibility when the experimental

computer became a contended resource by AEGIS group members. The use of DDT-86 (Dynamic Debugging Tool) to check memory locations and incrementally run the system proved to be the most important tool for testing and verifying the Signal Processor Interface Simulation when the assembly language routines were linked and the experimental computer was utilized.

IV. CONCLUSIONS

A. UTILIZATION AND CHANGABILITY OF THE SIGNAL PROCESSOR SIMULATION

The Signal Processor Interface Simulation is a tool that can be of significant value to future testing of the NPS AEGIS Group's AN/SPY-1A Radar Controller Model. The Target-Database system was developed to allow it's use not only with the Static Model as specifically implemented in this version, but also as the basis for a version to interface with a Dynamic Model. The individual functions that make up the total Signal Processor Simulation Sytem have been modularized to enhance the use and adaptability of this version to what ever future directions the Simulation efforts of the AEGIS Modeling Group may be. A comprehensive Users Manual has been provided in Appendix F for use with this version of the Signal Processor Simulation as a stand alone document. The only interfacing required for the members of the AEGIS Modeling Group with regards to this Signal Processor Simulation should be the substitution of MCORTEX "await" and "advance" primitives for those utilized in this version of the Static Model, and the possible restructuring of the address locations in common memory. It is recommended that any tester of the NPS SPY-1A Radar Controller Model first gain experience of the Signal Processor Simulation by running the Simulated SPY-1A Program "SPYTEST.CMD".

B. FUTURE ENHANCEMENTS AND DIRECTION FOR THE SPY-1A MODEL

The next logical step in the full implementation of the Signal Processor Interface Simulation is the further design and implementation of the Dynamic Model. The purpose of the Dynamic Model is to test the logic of the NPS SPY-1A Radar Controller Model. The Dynamic Model does not require the real-time performance of the Static Model, but must provide a comprehensive display of the active targets representing the Target-Database at each discrete time increment. The Target-Database system developed in this thesis should provide the basis for changing the structure of the Target-Database as the limits of the logical functions of the system are explored. However, the Target-Database has been purposefully designed for change should that be necessary in the implementation of the Dynamic Model. Previous thesis work by Boone [Ref. 3] should assist in the development of the Display module required for the Dynamic Model. Finally, the messages utilized (tables 58 and 8) for input and output from the Signal Processor Interface Simulation will require some attention. Specifically, the output message (table 8) needs to have some data from the input message to allow the SPY-1A Radar Controller Model to properly recognize and match input dwell commands with output data.

APPENDIX A

TARGET DATABASE PROGRAM LISTINGS

A. CONTROL.PLI

```
Prog Name  : CONTROL.PLI
Date       : May 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : This is the main program to control the
operation of the Signal Processor Simulation Target
Database functions and the Static Model functions.
*/

control:procedure options (main);

   declare
      create entry(pointer),
      delete entry(fixed,pointer),
      change entry(fixed,pointer),
      printlst entry(pointer,fixed),
      static entry;

   declare
      block fixed binary(7),
      init fixed decimal(2,1),
      init1 fixed,
      choice fixed binary(7),
      delta_t fixed decimal(2,1) EXTERNAL,
      endtime fixed decimal(4,1) EXTERNAL,
      time_in fixed,
      head_pointer;

entry errors */

   on error(1)
      begin;
         if block = 1 then do;
            put list(ascii(26),ascii(30));
            put skip list('invalid entry, try again...');
            go to start;
            end;
         if block = 2 then do;
            put list(ascii(26),ascii(30));
            put skip list('invalid entry,
                                must be integer 1-6...');
            go to menu;
            end;
         if block = 3 then do;
            put list(ascii(26),ascii(30));
            put skip list('invalid entry,
                                must be 1-',endtime,'...');
            go to branch;
            end;
      end;

   put list(ascii(26),ascii(30));  /*clear screen */
   put skip list('    ******* SIGNAL PROCESSOR SIMULATION
                                              *******');
   put skip list('                version 1.0 June 1983');
```

```
put skip(2);

start:

/* First determine what the time interval for display
updates and corresponding updates from the database will
be, as well as the length of the simulation */

block = 1;
put skip list('SYSTEM INITIATION: (see users manual)');
put skip list('How often do you want the database and
                                    the display updated?');
put skip list('    (delta_t range .1 to 1 seconds)');
put skip list('    (default is every .5 sec)');
put skip list('enter value or 0 for default: ');
get list(init);
if ((init>1)|(init<.1)) then signal error(1);
else delta_t = init;
put skip list('How many seconds do you want the
                                    simulation to run?');
put skip list('    (endtime range 1
                                to (delta_t * 8190))');
put skip list('    (default is 300 sec)');
put skip list('enter value or 0 for default: ');
get list(init1);
if ((init1>8190)|(init1<1)) then signal error(1);
else endtime = init1;

/*  Next the user will be placed in a interactive
environment where he can build track databases,
run simulation tests, and change the track database
as he desires */


do while('1'b);
   put list(ascii(26),ascii(30));/* clear screen */
   menu:
   block = 2;
   put skip list('          *** MAIN MENU ***');
   put skip(2);
   put skip list('What course of action do you wish?');
   put skip list('    (1)   CREATE a database of tracks');
   put skip list('          (you must do this first)');
   put skip list
         ('     (2)   DELETE a track from the database');
   put skip list
         ('     (3)   CHANGE a track on the database');
   put skip list
         ('     (4)    PRINT the current target list');
   put skip(1);
   put skip list
         ('After a database is satisfactory you may:');
   put skip list('    (5)   RUN a simulation');
   put skip list
('          (insure the rest of the SPY-1 Model is setup)');
   put skip list
    ('    (6)   QUIT and return to the operating system');
   put skip list('(enter 1-6 and <cr>):');
   get list(choice);
   if ((choice<1)|(choice>6)) then signal error(1);

   branch:
   block = 3;
   if choice = 1 then call create(head);
   if choice = 2 then do;
      put skip list
      ('At what time do you want to delete a target? ');
      get list(time_in);
      if ((time_in<1)|(time_in>endtime))
```

```
                                        then signal error(1);
      call delete(time_in,head);
   end;
   if choice = 3 then do;
      put skip list
      ('At what time do you want to change a target? ');
      get list(time_in);
      if ((time_in<1)|(time_in>endtime))
                                        then signal error(1);
      call change(time_in,head);
   end;
   if choice = 4 then call printlst(head,time_in);
   if choice = 5 then call static;
   if choice = 6 then do;
      put skip(2) list
                  ('          *** END OF SIMULATION ***');
      revert error(1);
      stop;
   end;
end; /* while */
```

B. CREATE.PLI

```
end control;
Prog Name   : CREATE.PLI
Date        : May 83
Written by  : Todd B. Kersh
For         : Thesis (AEGIS Modeling Group)
Advisor     : Professor Kodres
Purpose     : This module is part of the Target
Database package of functions.
*/

create: procedure (head);

   /* Global declarations */

   %include 'dbase.dcl';

   /* Local declarations */

   declare
      bld_database_entry (fixed,pointer),
      i fixed binary(7),
      cont character(1) static init('Y'),
      block fixed binary(7),
      tgtnum fixed binary(7) static init(0) external,
      xrange float,
      yrange float,
      xvel float,
      yvel float,
      xacel float,
      yacel float,
      alt float,
      tgteq fixed binary(7);

entry errrors */

on error(1)
   begin;
     put skip list('ENTRY ERROR, TRY AGAIN...');
     if block = 1 then
        go to retry;
     if block = 2 then
        go to again;
   end;

   put skip list('=== CREATE TARGETS MODULE ===');

   /* Initiate the target list */

      allocate target_set(tgt_mkr);
      tgt_ptr = tgt_mkr;
      head = tgt_mkr;
      tgt_ptr->num = 0;  /* this is the header node */
      allocate target_set(tgt_mkr);
      tgt_ptr->next_ptr = tgt_mkr;
      tgt_ptr = tgt_mkr;

   /* Create the list of targets to be simulated */

   do while ( cont = 'Y');
      tgtnum = tgtnum + 1;
      tgt_ptr->num = tgtnum;
      retry:
      block = 1;
      put skip list('Initiate target#',tgtnum);

      /* Assign the target parameters */
```

```
      put skip list
                ('   Parametric Equations? (1,2,3,or4): ');
      get list(tgteq);
      if ((tgteq<1)|(tgteq>4)) then signal error(1);

      put skip list('   X_range (a)? (-256,+256)nm: ');
      get list(xrange);
      if ((xrange<-256)|(xrange>256)) then signal error(1);

      put skip list('   Y_range (u)? (-256,+256)nm: ');
      get list(yrange);
      if ((yrange<-256)|(yrange>256)) then signal error(1);

      put skip list('   X_velocity (b)? (-32,+32)m/sec: ');
      get list(xvel);
      if ((xvel<-32)|(xvel>32)) then signal error(1);

      put skip list('   Y_velocity (v)? (-32,+32)m/sec: ');
      get list(yvel);
      if ((yvel<-32)|(yvel>32)) then signal error(1);

      put skip list
('    X_acceleration (c)? (-.015625,+.015625)m/sec/sec: ');
      get list(xacel);
      if ((xacel<-.015625)|(xacel>.015625))
                                        then signal error(1);

      put skip list
('    Y_acceleration (w)? (-.015625,+.015625)m/sec/sec: ');
      get list(yacel);
      if ((yacel<-.015625)|(yacel>.015625))
                                        then signal error(1);

      put skip list('   Z_altitude (d)? (0,20,000)ft: ');
      get list(alt);
      if ((alt<0)|(alt>20000)) then signal error(1);

      tgt_ptr->eq = tgteq;
      tgt_ptr->a = xrange;
      tgt_ptr->b = xvel;
      tgt_ptr->c = xacel;
      tgt_ptr->d = alt;
      tgt_ptr->u = yrange;
      tgt_ptr->v = yvel;
      tgt_ptr->w = yacel;

      /* Determine if more targets are to be created */

      again:
      block = 2;
      put skip(2) list('create more targets?(Y or N): ');
      get list(cont);
      if cont = 'y' then cont = 'Y';
      if ((cont = 'Y')&(tgtnum^=56)) then do;
         allocate target set(tgt_mkr);
         tgt_ptr->next_ptr = tgt_mkr;
         tgt_ptr = tgt_mkr;
         end;
      if tgtnum = 56 then do;
         put skip list('TARGET LIST IS FULL...');
         cont = 'N';
         end;
   end; /*while cont */
   cont = 'Y';

   /* Complete the linked list */

   tgt_ptr->next_ptr = null;
```

```
    tgt_ptr = head;
    tgt_mkr = head;

    /* Build the Remex Data Warehouse database. */

    put skip list('BUILDING DATABASE...');
    call bld_database(1,head);
    revert error(1);

end create;
```

C. DELETE.PLI

```
Prog Name   : DELETE.PLI
Date        : May 83
Written by  : Todd B. Kersh
For         : Thesis (AEGIS Modeling Group)
Advisor     : Professor Kodres
Purpose     : This module is part of the Target
Database package of functions. It deletes targets
from the Target List.
*/

delete: procedure (time_in,head);
   %replace
      true by '1'b,
      false by '0'b;

   /* Global declarations */

   %Include 'dbase.dcl';

   /* Local declarations */

   declare
      bld_database entry (fixed,pointer),
      found bit(1) static init(false),
      time_in fixed,
      tgtnum fixed binary(7) external,
      tgt fixed binary(7),
      cont character(1) static init('Y');

   /* This exception handler will take care of
   all user input errors */

   on error(1)
      begin;
         put skip list('ENTRY ERROR, TRY AGAIN ...') ;
         go to retry;
      end;
   tgt = 0;

   put skip list('=== DELETE TARGETS MODULE ===');

   /* This will initialize the Target linked list to the
   correct memory space */

   tgt_ptr = head->next_ptr;
   tgt_mkr = head;

   /* This will delete the desired node from the
   target linked list */

   do while( cont = 'Y');
      retry:
      put skip list
                  ('    What target do you wish to delete? ');
      put skip list
                  ('      (tgt. num. range 1-',tgtnum,'): ');
      get list(tgt);
      if ((tgt<1)|(tgt>tgtnum)) then signal error(1);

      do while(found = false);
       if tgt_ptr->num = tgt then do;
         tgt_mkr->next_ptr = tgt_ptr->next_ptr;
         tgt_ptr->next_ptr = null;
         free tgt_ptr->target;
         tgt_ptr = head->next_ptr;
         tgt_mkr = head;
```

```
            found = true;
         end;
         else do;
            tgt_mkr = tgt_ptr;
            tgt_ptr = tgt_mkr->next_ptr;
            if tgt_ptr = null then do;
               put skip list
                        ('ERROR : target number not found');
               tgt_ptr = head->next_ptr;
               tgt_mkr = head;
               found = true;
               end;
         end;
        end; /* while */
        found = false;

        put skip list('continue (Y/N)? ');
        get list(cont);
        if cont = 'y' then cont = 'Y';
    end; /*while*/
    cont = 'Y';

    put skip(2) list('BUILDING NEW DATABASE...');
    call bld_database(time_in,head);
    revert error(1);

end delete;
```

D. CHANGE.PLI

```
Prog Name  : CHANGE.PLI
Date       : May 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : This module is part of the Target
Database package of functions. It changes data
on the Target List.
*/

change: procedure (time_in,head);

   %replace
      true by '1'b,
      false by '0'b;

   /* Global Declarations */

   %include 'dbase.dcl';

   /* Local Declarations */

   declare
      bld_database_entry (fixed,pointer),
      time_in fixed,
      more bit(1) static init(true),
      tgtnum fixed binary(7) external,
      tgt fixed binary(7),
      (chg1,chg2) fixed binary(7),
      (chg3,chg4,chg5,chg6,chg7,chg8,chg9) float,
      do1 bit(1) static init(false),
      block fixed binary(7),
      cont character(1) static init('Y');

   /* This exception handler will take care of all
   user input errors */

   on error(1)
      begin;
         put skip list('ENTRY ERROR, TRY AGAIN...');
         if block = 1 then
            go to try1;
         if block = 2 then
            go to try2;
      end;

   put skip list('=== CHANGE TARGETS MODULE ===');

   /* First, query the user about the changes to be made */

   do while(cont = 'Y');
      try1:
      block = 1;
      put skip list
      ('   What is the target number you wish to change?');
      put skip list
                  ('      (tgt. num. range 1-',tgtnum,'): ');
      get list(tgt);
      if ((tgt<1)|(tgt>tgtnum)) then signal error(1);

      put skip list('   What data item is to be changed?');
      put skip list('      (1) parametric equation');
      put skip list('      (2) equation parameters');
      get skip list(chg1);
      if ((chg1<1)|(chg1>2)) then signal error(1);
```

```
   if chg1 = 1 then do;
      do1 = true;
      put skip list
           ('    What is the new equation number (1-4)?');
      get list(chg2);
      if ((chg2<1)|(chg2>4)) then signal error(1);
   end;
   else do;
      try2:
      block = 2;
      do1 = false;

      put skip list('    What are the new parameters:');
      put skip list
                ('      X_range (a)? (-256,+256)nm: ');
      get list(chg3);
      if ((chg3<-256)|(chg3>256)) then signal error(1);

      put skip list
                ('      Y_range (u)? (-256,+256)nm: ');
      get list(chg4);
      if ((chg4<-256)|(chg4>256)) then signal error(1);

      put skip list
           ('      X_velocity (b)? (-32,+32)m/sec: ');
      get list(chg5);
      if ((chg5<-32)|(chg5>32)) then signal error(1);

      put skip list
           ('      Y_velocity (v)? (-32,+32)m/sec: ');
      get list(chg6);
      if ((chg6<-32)|(chg6>32)) then signal error(1);

      put skip list
('      X_accel.   (c)? (-.015625,+.015625)m/sec/sec: ');
      get list(chg7);
      if ((chg7<-.015625)|(chg7>.015624))
                                   then signal error(1);

      put skip list
('      Y_accel.   (w)? (-.015625,+.015625)m/sec/sec: ');
      get list(chg8);
      if ((chg8<-.015625)|(chg8>.015625))
                                   then signal error(1);

      put skip list('      Z_alt.   (d)? (0;20,000)ft: ');
      get list(chg9);
      if ((chg9<0)|(chg9>20000)) then signal error(1);
   end;

   tgt_ptr = head->next_ptr;
   tgt_mkr = head;

   /* Now this will find the desired node, and make the
   requested changes on the target data list */

   do while(more = true);
      if tgt_ptr->num = tgt then do;
         if do1 = true then tgt_ptr->eq = chg2;
         else do;
            tgt_ptr->a = chg3;
            tgt_ptr->b = chg5;
            tgt_ptr->c = chg7;
            tgt_ptr->u = chg4;
            tgt_ptr->v = chg6;
            tgt_ptr->w = chg8;
            tgt_ptr->d = chg9;
         end;
         more = false;
```

```
            end;
            else do;
               tgt_mkr = tgt_ptr;
               tgt_ptr = tgt_mkr->next_ptr;
               if tgt_ptr = null then do;
                  put skip list
                        ('ERROR : target number not found');
                  tgt_ptr = head->next_ptr;
                  tgt_mkr = head;
                  more = false;
               end;
            end;
         end; /* while */
         more = true;

         put skip(2) list
                  ('Do you wish to change another target?');
         put skip list('  (Y/N): ');
         get list(cont);
         if cont = 'y' then cont = 'Y';

      end; /* while */
      cont = 'Y';

      put skip(2) list('UPDATING CHANGED DATABASE...');
      call bld_database(time_in,head);
      revert error(1);

end change;
```

E. BLDDBASE.PLI

```
Prog Name  : BLDDBASE.PLI
Date       : May 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : This is the module the builds the
database in the Remex Data Warehouse after the
Target List has been created or modified.
*/

bld_database: procedure (time_in,head);
   %replace
      true by '1'b,
      false by '0'b;

   /* Global Declarations */

   %include 'dbase.dcl';

   /* Local Declarations */
   declare
      timeend fixed,
      time_in fixed binary(15),
      t fixed binary(15),
      trkfull char(1) static init('N'),
      i fixed;

   /* This main procedure uses subprocedures to build
   the database of table-8 structures in the Remex Data
   Warehouse */

   t = time_in;
   timeend = trunc(endtime/delta_t);
   do i = time_in to timeend;
      call bld_msg_buffer(head,t);
      call load_rdw(t);
      t = t + 1;
      if trkfull = 'Y' then return;
   end;

/* This procedure will create the Signal Processor
Interface Simulation output message to the Track
Processor Module for each node of the target_list,
and store them in a buffer. */

bld_msg_buffer: procedure (head,time_in);

   declare

  /* The Buffer contains all the track tables at time_in */

      1 Buffer static,

         /* Table 8; interfaces between the beam
         stabilization process and the track process */

         2 B_to_P_tabl(57),
            3 x_sub_s fixed binary(15) init(0),
            3 y_sub_s fixed binary(15) init(0),
            3 z_sub_s fixed binary(15) init(0),
            3 face_idx fixed binary(7) init(0),
            3 dwl_idx fixed binary(7) init(0),
            3 trk_num fixed binary(7) init(0);

   declare
      bld_buff entry (1,2 pointer,
```

```
                    2 bit(16),
                    2 bit(16));

declare
   time_in fixed binary(15);
   head pointer;


declare
   more bit(1) static init(true),
   ctr fixed binary(7),
   equ fixed binary(7),
   (x,y,z) float;

declare
   1 parablk static,
      2 sourcebuff pointer,
      2 destbuff bit(16) init('5500'b4),
      2 segaddr bit(16) init('e000'b4);

ctr = 1;

/* First get the node of the target data list and
   extract the data needed to generate the items
   on tbl 8 */

  tgt_ptr = head->next_ptr;
  tgt_mkr = tgt_ptr;

do while(more = true);


  buffer.b_to_p_tabl(ctr).trk_num = tgt_ptr->num;
   equ = tgt_ptr->eq;

   /* Derive values for target positions x,y,and z
   at time_in for the specified parametric equation */

   if equ = 1 then do;
      x=tgt_ptr->a + tgt_ptr->b*time_in
                          tgt_ptr=>c*time_in*time_in;
      y=tgt_ptr->u + tgt_ptr->v*time_in
                          tgt_ptr=>w*time_in*time_in;
      z=tgt_ptr->d;
   end;

   if equ = 2 then do;
      x=tgt_ptr->a + tgt_ptr->b*time_in
                          tgt_ptr=>c*time_in*time_in;
      y=tgt_ptr->u + tgt_ptr->v*sin(tgt_ptr->w*time_in);
      z=tgt_ptr->d;
   end;

   if equ = 3 then do;
      x=tgt_ptr->a + tgt_ptr->b*cos(tgt_ptr->c*time_in);
      y=tgt_ptr->u + tgt_ptr->v*time_in
                          tgt_ptr=>w*time_in*time_in;
      z=tgt_ptr->d;
   end;

   if equ = 4 then do;
      x=tgt_ptr->a + tgt_ptr->b*cos(tgt_ptr->c*time_in);
      y=tgt_ptr->u + tgt_ptr->v*sin(tgt_ptr->w*time_in);
      z=tgt_ptr->d;
   end;

      buffer.b_to_p_tabl(ctr).x_sub_s = x;
      buffer.b_to_p_tabl(ctr).y_sub_s = y;
      buffer.b_to_p_tabl(ctr).z_sub_s = z;
```

```
         /* Set up tc look at the next target */

         ctr = ctr + 1;
         tgt_ptr = tgt_mkr->next_ptr;
         tgt_mkr = tgt_ptr;
         if tgt_ptr = null then more = false;
   end; /*dc while*/
   more = true;
   tgt_ptr = head;
   tgt_mkr = tgt_ptr;

   /* This will transfer the buffer structure to the
   common memory board buffer location for transfer
   to the REMEX. */

   parablk.scurcebuff = addr(buffer);
   call hld_buff(b_ptr);

end hld_msg_buffer;

/*  This procedure will cause the REMEX Data Ware-
house to load the contents of the buffer into the
next sector on the RDW hard disk. */

load_rdw: procedure(time_in);

   declare
      time_in fixed,
      send_mess entry,
   build_cmd_mess entry (bit(16), fixed binary(15),
                         fixed binary(15),
                         fixed binary(7), fixed binary(7),
                         bit(16), bit(16),
                         fixed binary(15));
   declare
      status fixed binary(15) static init(0),
      sect fixed binary(7),
      word_count fixed binary(15) static init(256),
      mem bit(16) static init('5500'b4),
      msb bit(16) static init('000e'b4),
      track fixed binary(15),
      head fixed binary(7),
      rdw_read bit(16) static init('1020'b4);
               /* "1020" means the REMEX will write
              from the com.mem. buffer to the hard disk.*/

   head = 0;  /* this sets head to "D" drive */

  /* This determines the sector based on 39 sectors/track */

   sect = 1 + mod(time_in,40);

  /* This determines the track */

   track = 1 + trunc(time_in/39);

   /* need except. hndler for TRACK >210 */

   if track > 210 then do;
      put skip list('The database store is full.');
      put skip list
       ('  This run is recorded as ',time_in,' delta_ts.');
      put skip list
('  To create a longer run, change the value of delta_t.');
      trkfull = 'Y';
      return;
      end;
```

```
    /* This procedure builds the command message required
    for the REMEX Data Warehcuse to read the data tables
    located in the buffer corresponding to the value
    of "time_in". */

    call build_cmd_mess
       (rdw_read,status,track,head,sect,mem,msb,word_count);

    /* The procedure sends the command message to the Remex
    Data Warehouse to perform the required read operation */

    call send_mess;

end load_rdw;

end bld_database;
```

F. PRINTLST.PLI

```
Prog Name   : PRINTLST.PLI
Date        : May 83
Written by  : Todd B. Kersh
For         : Thesis (AEGIS Modeling Group)
Advisor     : Professor Kodres
Purpose     : This module is a diagnostic tool for
the user to keep a record of the flow of the Target
List as changes are made at each delta_t, as a Target
Database is constructed for a Static Model run.
*/

printlst: procedure (head,time);
   %include 'dbase.dcl';

   declare
      prt fixed bin(7),
      time fixed,
      (head,ap,bp) pointer;

   put skip list('=== PRINT TARGET LIST ===');
   retry:
   put skip list('To get a print out, turn on printer,
 put skip list('type <ctrl> P, and then type 0<return>.');
   put skip list('    Else, just type 0<return>.');
   get list(prt);
   if prt ^= 0 then go to retry;

   put skip(2) list('TARGET LINKED LIST at time = ',time);
   ap = head;
   bp = ap;
   ap = bp->next_ptr;

   do while(ap^=null);
      bp = ap;
      put skip(2);
      put skip list('TGT  : ',ap->num);
      put skip list('   EQ : ',ap->eq);
      put skip list('    a : ',ap->a);
      put skip list('    b : ',ap->b);
      put skip list('    c : ',ap->c);
      put skip list('    u : ',ap->u);
      put skip list('    v : ',ap->v);
      put skip list('    w : ',ap->w);
      put skip list('    d : ',ap->d);
      ap = bp->next_ptr;
   end; /* while */

   ap = head;
   bp = head;

end printlst;
```

G. DBASE.DCL

```
/*
Prog Name  : DBASE.DCL
Date       : May 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : These are the global declarations
for the Target Database.
*/


        declare
                endtime fixed decimal(4,1) external,
                delta_t fixed decimal(2,1) external;
        declare
                head pointer,
                (tgt_mkr,tgt_ptr) pointer,
                1 target based,
                        2 num fixed binary(7),
                        2 eq fixed binary(7),
                        2 para,
                                3 a float,
                                3 b float,
                                3 c float,
                                3 u float,
                                3 v float,
                                3 w float,
                                3 d float,
                                3 e float,
                                3 f float,
                        2 next_ptr pointer;
```

H. BLDBUFF.A86

```
;Prog Name   :  BLDBUFF.A86
;Date        :  4 June 83
;Written by :  Todd E. Kersh
;For         :  Thesis (AEGIS Modeling Group)
;Advisor     :  Professor Kodres
;Purpose     :  This routine will transfer a 256 word
;  buffer from SBC private memory to the com.
;  memory buffer starting at E000:5100.  The
;  parameter passed is a parameter block containing
;  a pointer to the buffer on SBC, and the base
;  and offset to the common mem. buffer.
;
;=============================================
;    Code Segment
;=============================================
; This routine assumes parameters as follows:
;   para1    parameter block consisting of
;                3 words.
;
cseg

bld_buff:
    push ax
    push si
    push di
    push cx
    push es
    mov si,[bx]    ; get location of buff1
    mov si,[si]    ;   from para. passed
    mov di,[bx]    ; assign location of buff2
    les di,2[di]
    mov cx,256       ; assign no. of words to move
move_words:
    mov ax,[bx]    ; load word from source
    mov es:[di],ax   ; store word into com.mem. buffer
                       ; adjust pointers.
    inc si
    inc si
    inc di
    inc di
    loop  move_words ; loop if not done
    pop es
    pop cx
    pop di
    pop si
    pop ax
    ret
end
```

APPENDIX B

STATIC MODEL PROGRAM LISTINGS

A. STATIC.PLI

```
Prog Name   : STATIC.PLI
Date        : 8 June 83
Written by  : Todd B. Kersh
For         : Thesis (AEGIS Modeling Group)
Advisor     : Professor Kodres
Purpose     : This module controls the operation for
the RCP Static Model.
*/

static: procedure;

   declare
      load_buffer entry(fixed),
      xfer_msg entry,
      advance_evc1 entry,
      display_entry(fixed),
      await entry (fixed binary(15));

   declare
   start fixed binary(7),
      thrshcld fixed binary(15) static init(1),
      endtime fixed external,
      item fixed binary(7),
      evcvalue fixed binary(15) static init(0) external,
      time fixed;

   /*  This exception handler will take care of all
   user input errors. */

   on error(1)
      begin;
         put skip list('ENTRY ERROR, TRY AGAIN...');
         go to retry;
      end;

   put skip(ascii(26),ascii(30)); /* clr. screen */
   put skip list('    === RSP STATIC MODEL ===');
   put skip list('      version 1.0 June 83');
   put skip(2);
   put skip list('At this point you should have created
                  a database and are now ready to run');
   put skip list('your test of the NPS SPY-1A Model.');
   put skip(2);
   retry:
   put skip list(' === STATIC MODEL MENU ===');
   put skip list('(1) TEST run the simulation');
   put skip list('(2) QUIT and return to main menu');
   put skip list('enter 1-2 and <cr>: ');
   get list(item);
   if ((item<1)|(item>2)) then signal error(1);

   if item = 1 then do;
      put skip list('Load SPYTEST.CMD from another
                     system CRI/SBC.');
      put skip list('When complete, enter 0<cr>
                     to begin => ');
```

```
      get list(start);
      if start^=0 then signal error(1);

      do time = 1 to endtime;
         call await(threshold);
         threshold = threshold + 1;
         call load_buffer(time);
         call xfer_msg;
         call advance_evc1;
         call display(time);
      end;
   end;
   else return;

   revert error(1);

end static;
```

B. AWAIT.A86

```
;
;Prog Name   : AWAIT.A86
;Date        : 8 June 83
;Written by : Todd B. Kersh
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This module checks to see if a msg has been
;written to 0E000:5614 of common memory by the Radar
;Scheduler.
;
;=============================================
;       DATA
;=============================================
;
;
;=============================================
;      CODE
;=============================================
;
cseg
public   await

await:
   push ax
   push di
   push si
   push es
   mov ax,0e000h          ; get common mem. base
   mov es,ax              ; assign to eseg base
   mov di,06020h          ; point to evc addrs.
   mov si,[bx]          ; get threshold
   lods ax                ; put it in ax reg.
poll:
   cmp ax,es:[di]     ; compare evc to thr
   jnz  poll              ; if no new msg, wait
   pop es                 ; else return
   pop si
   pop di
   pop ax
   ret
end
```

C. LOADBUFF.PLI

```
Prog Name  : LOADBUFF.PLI
Date       : 31 May 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : This module is part of the Static Model
and will extract the proper sector of output msgs (e.g.
tbl_8s) from the database on the Remex DW, based on the
current value of delta_t, and place the data in the
common memory buffer.
*/

load_buffer: procedure(time_in);

   declare
      time_in fixed,
      send_mess entry,
  build_cmd_mess entry (bit(16), fixed binary(15),
                        fixed binary(15),
                        fixed binary(7), fixed binary(7),
                        bit(16), bit(16),
                        fixed binary(15));
   declare
      status fixed binary(15) static init(0),
      sect fixed binary(7),
      word_count fixed binary(15) static init(256),
      mem bit(16) static init('5500'b4),
      msb bit(16) static init('000e'b4),
      track fixed binary(15),
      head fixed binary(7),
      rdw_wrt bit(16) static init('1010'b4);
          /* "1010" means the REMEX will read the
             hard disk and write to com.mem. buffer. */

   head = 0; /* this sets head to "D" drive */

  /* This determines the sector based on 39 sectors/track */

   sect = 1 + mod(time_in,40);

   /* This determines the track */

   track = 1 + trunc(time_in/39);

   /* Max tracks available on the REMEX is 210
   therefor, to prevent running out of memory...*/

   if track > 210 then do;
      put skip list('The database store is full.');
      put skip list
      ('  This run is recorded as ',time_in,' delta_ts.');
      put skip list
('  To create a longer run, change the value of delta_t.');
      return;
      end;

   /* This procedure builds the command message
   required for the Remex Data Warehouse to write the
   data tables located in the buffer corresponding
   to the value of 'time_in' */

   call build_cmd_mess
       (rdw_wrt,status,track,head,sect,mem,msb,word_count);

   /* The procedure sends the command message to the
   Remex Data Warehouse to perform the required
```

```
   write operation */
   call send_mess;
end load_buffer;
```

D. XFER.A86

```
;
;Prog Name  : XFER.A86
;Date       : 8 June 83
;Written by : Todd B. Kersh
;For        : Thesis (AEGIS Modeling Group)
;Advisor    : Professor Kodres
;Purpose    : This module will transfer a output msg.from
;the common memory buffer to the common memory location
;0E000:5646 to be read by the Track Processing Module.
;
;=============================================
;     Data
;=============================================
;
;
;=============================================
;     Code
;=============================================
;
cseg
public xfer_msg

xfer_msg:
    push di
    push si
    push es
    push ds
    push ax
    pushf
    mov ax,0e000h                ;get common mem. base
    mov es,ax                    ;assign to eseg
    mov si,C5500h
    mov di,06055h
    mov cx,5                     ;set loop ctr to
                                 ; pass 5 words
                                 ; (one tbl_8).
move_msg:
    mov ax,es:[si]              ;load word from buffer
    mov es:[di],ax              ;store word into msg.
    inc si                       ;get next word loc.
    inc si
    inc di                       ;get next word loc.
    inc di
    loop move_msg                ;loop until done
    popf
    pop ax
    pop ds
    pop es
    pop si
    pop di
    ret
end
```

E. DISPLAY.PLI

```
Prog Name   : DISPLAY.PLI
Date        : 8 June 83
Written by  : Todd B. Kersh
For         : Thesis (AEGIS Modeling Group)
Advisor     : Professor Kodres
Purpose     : This will display the status of the test
simulation for the RSP Static Model, and allow the
user to make measurements to determine the speed of the
NPS SPY-1A Model execution.
*/

display: procedure (time);

   declare
      time fixed,
      endtime fixed external;

   /*put list(ascii(26),ascii(30));*/
   put skip list('   === RSP STATIC MODEL SIMULATION ===');
   put skip(2);
   put skip list('TIME:',time,'  ENDTIME:',endtime);

end display;
```

F. ADV1EVC.A86

```
;
;Prog Name : ADV1EVC.A86
;Date      : 8 June 83
;Written by: Todd B. Kersh
;For       : Thesis (AEGIS Modelling Group)
;Advisor   : Professor Kodres
;Purpose   : This module will simulate the Radar
;Signal Processor sending a new data msg to the
;SPY-1A Controller Model.
;
;=============================================
;   Data
;=============================================
;
;
;=============================================
;   Code
;=============================================
;
cseg
public  advance_evc1

advance_evc1:
     push es
     push di
     push ax
     mov ax,0e000h
     mov es,ax              ; get com. mem. base to eseg
     mov di,06055h          ; point to evc1
     mov ax,es:[di]     ; get value
     inc ax                 ; increment value
     mov es:[di],ax     ;store evc1
     pop ax
     pop di
     pop es
     ret
end
```

APPENDIX C

COMMON ASSEMBLY LANGUAGE LISTINGS

A. BLDMESSG.A86

```
;
;Prog Name   : BLDMESSG.A86
;Date        : May 83
;Written by  : Todd B. Kersh
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Kodres
;Purpose     : This primitive module is a general
;command msg. passing routine to the Remex Data
;Warehouse, to be used for both Write and Read
;operations.  It expects to get parameters as
;follows from the calling PLI program:
;    build_cmd_mess(word 0, word 1, word 2,
;                   word 3 high byte, word 3
;                   low byte, word 4, word 5,
;                   word 6)
;
;=============================================
;       Data Segment
;=============================================
DSEG
        COMMEM          EQU     0E000H
ESEG
        ORG 5400H
PUBLIC  STATUS
        MODIFIERS       RW      1
        STATUS          RW      1
        TRACK_NO        RW      1
        HEAD_SECT       RW      1
        MEM_ADDR        RW      1
        MSB             RW      1
        WORD_CNT        RW      1
;=============================================
;       Code Segment
;=============================================
CSEG
PUBLIC  BUILD_CMD_MESS

BUILD_CMD_MESS:
        PUSH ES
        PUSH CX
        PUSH SI
        PUSH BX
        PUSH AX
        PUSHF
     MOV AX,COMMEM
     MOV ES,AX
        CLD
        MOV SI,[BX]     ; set source index to point
                        ; to 1st parameter.
        LODS AX         ; AX = para 1, SI incremented
                        ; to point to next parameter.
        MOV MODIFIERS,AX
        ADD BX,02H      ;point to next parameter address
        MOV SI,[BX]     ;set source index to
                        ;point to next para.
        LODS AX
```

```
        MCV STATUS,AX
        ADD BX,02H          ;point to next parameter address
        MCV SI,[BX]       ;set source index to point
                            ;to next para.
        LCDS AX
        MCV TRACK_NO, AX
        ADD BX,02H          ; point to next parameter address
        MOV SI,[BX]      ;set index to point to next para.
        LCDS AL             ; get byte para in al reg.
        MOV AH,AL           ; move al to ah
        ADD BX,02H
        MCV SI,[BX]
        LCDS AL             ;word is now complete for movement
        MOV HEAD_SECT,AX
        ADD BX,02H          ;point to next parameter address
        MOV SI,[BX]       ;set source index to
                            ;point to next para.
        LODS AX
        MCV MEM_ADDR,AX
        ADD BX,02H          ;point to next parameter address
        MCV SI,[BX]       ;set source index to
                            ;point to next para.
        LCDS AX
        MCV MSB,AX
        ADD BX,02H          ;point to next parameter address
        MOV SI,[BX]       ;set source index to
                            ;point to next para.
        LCDS AX
        MCV WORD_CNT,AX
        POPF
        POP AX
        POP BX
        POP SI
        POP CX
        POP ES
    RET
END
```

B. SNDMESS1.A86

```
;
;Prog Name  : SNDMESS1.A86
;Date       : May 83
;Written by : Todd B. Kersh
;For        : Thesis (AEGIS Modeling Group)
;Advisor    : Professor Kodres
;Purpose    : This primitive module sends the command
;message located in common memory at 0E000:5000 to the
;Remex Data Warehouse for execution.  It checks the
;Status Word in the msg. for successful msg completion.
;
;=============================================================
;           Data Segment
;=============================================================
;
DSEG
     RDW_ERROR          DB      1
     RDW_RESULT         DB      1
     RDW_DIR            DB      1
     SUCCESS            EQU     1       ;code for opn success
     FAILURE            EQU     0       ;code for opn failure
     RDW_READY          EQU     08H
     TRIES              EQU     05       ;constant
     ;
     ;  RDW interface controller ports ==>
     ;
     RDW_CMD_REG        EQU     70H
     RDW_STATUS_REG     EQU     71H
     RDW_ADDR_LO        EQU     72H
     RDW_ADDR_HI        EQU     73H

ESEG
EXTRN     STATUS:WORD
;
;=============================================================
;           Code Segment
;=============================================================
;
CSEG
PUBLIC SEND_MESS:

     SEND_MESS:
          PUSHF
          PUSH AX
          PUSH ES
          PUSH CX
          MOV AX,0E000H
          MOV ES,AX
          MOV CX,TRIES      ;init. loop counter
     SEND_RDW_MESS:
          IN AL,RDW_STATUS_REG
          AND AL,RDW_READY      ;check if interface ready
                                ;to process cmd packet...
          CMP AL,08H           ;ready?
          JNE SEND_RDW_MESS     ;if not repeat
          MOV AL,1CH
          OUT RDW_CMD_REG,AL     ;load extended address
          MOV AX,05400H             ;offset of packet
          OUT RDW_ADDR_LO,AL     ;transfer low byte
          MOV AL,AH
          OUT RDW_ADDR_HI,AL     ;transfer high byte
     CHECK_RDW_RESULT:
          MOV AX,STATUS           ;read status word
          CMP AX,SUCCESS         ;check for success
          JE  RDW_SUCCESS_READ
          CMP AX,FAILURE          check for failure
```

```
            JNE RDW_RETRY
            JMPS CHECK_RDW_RESULT
      RDW_RETRY:
            MOV RDW_ERROR,AL      ;save error code
            MOV STATUS,0            ;clear status word
            LOOP SEND_RDW_MESS      ;loop if counter <> 0
            MOV RDW_RESULT,0FFH      ;return failure code
            JMPS RDW_EXECUTE_RET
     RDW_SUCCESS_READ:
            MOV RDW_RESULT,00H      ;return success code
      RDW_EXECUTE_RET:
            POP CX
            POP ES
            POP AX
            POPF
            RET

END
```

APPENDIX D

SPY-1A MODEL SIMULATION PROGRAM LISTINGS

A. SPYTEST.PLI

```
/*
Prog Name  : SPYTEST.PLI
Date       : 8 June 83
Written by : Todd B. Kersh
For        : Thesis (AEGIS Modeling Group)
Advisor    : Professor Kodres
Purpose    : This is a test module that simulates the
operation of the NPS SPY-1A Model.  It will update the
tabl_58 data in common memory upon the event count update
from the Static Model, after a delay loop that simulates
the operation of the Track Processor and Radar Scheduler.
*/

spy_test: procedure;

   declare
      init entry,
      advance_evc2 entry,
      threshold fixed bin(15) static init(1),
      await1 entry (fixed binary(15)),
      i fixed binary (15);

   /* This will initiate the eventcounts to 0 */

   call init;

   do while ('1'b);
      call advance_evc2; /* This simulates sending a new
                            dwell cmd. msg. */
      call await1(threshold); /* wait for results
                                 e.g. tbl_8 */

      /* This is a delay loop simulating SPY-1A Model
      processing time. */

      threshold = threshold + 1;

      do i = 1 to 50;
         put skip list('SPY-1 IS PROCESSING DATA...');
      end;

   end; /* while */

end spy_test;
```

B. INIT.A86

```
;
;Prog Name  : INIT.A86
;Date       : 19 June 83
;Written by : Todd B. Kersh
;For        : Thesis (AEGIS Modeling Group)
;Advisor    : Professor Kodres
;Purpose    : This module initiates the
;memory locations for the eventcounts to 0.
;
;=============================================
;     Data
;=============================================
;
dseg

eseg
public  evc1,evc2
        org 06020h
        evc2  rw   1
        evc1  rw   1
;
;=============================================
;     Code
;=============================================
;
cseg
public  init

init:
    push ax
    push es
    mov  ax,0e000h
    mov  es,ax              ;set exeg base to com. mem
    mov  ax,0               ;set ax to 0
    mov  evc1,ax            ;set eventcounts to 0
    mov  evc2,ax
    pop es
    pop ax
    ret
end
```

C. AWAIT1.A86

```
;
;Prog Name  : AWAIT1.A86
;Date       : 8 June 1983
;Written by : Todd B. Kersh
;For        : Thesis
;Advisor    : Professor Kodres
;Purpose    : This module checks to see if an evc. has
;been updated at 0E000:5616 of common memory by the
;Radar Signal Processor Simulator.
;
;=============================================
;   Data
;=============================================
;
dseg

eseg
extern  evc1:word
;
;=============================================
;   Code
;=============================================
;
cseg
public  await1

await1:
     push si
     push ax
     push es
     mov ax,0e000h
     mov es,ax              ; get com. mem base in eseg
     mov si,[bx]          ; get parameter - threshold
     lds ax                 ; load it in ax reg.
poll:
     cmp ax,evc1            ; compare evc to threshold
     jnz poll               ; if no new message sent, wait
     pop es                 ; else, return
     pop ax
     pop si
     ret
end
```

D. ADV2EVC.A86

```
;
;Prog Name  :  ADV2EVC.A86
;Date       :  19 June 83
;Written by :  Todd E. Kersh
;For        :  Thesis (AEGIS Modeling Group)
;Advisor    :  Professor Kodres
;Purpose    :  This module will simulate the Radar
;Scheduler sending a new dwell command to the RSP.
;
;=============================================
;    Data
;=============================================
;
dseg

eseg
extrn  evc2:word
;
;=============================================
;   Code
;=============================================
;
cseg
public   Advance_evc2

advance_evc2:
    push es
    push ax
    mov  ax,0e000h
    mov  es,ax           ; get addr. of comm.mem. base
                         ;    in eseg
    inc evc2; advance event count
    pop ax
    pop es
    ret
end
```

APPENDIX E
OBJECT-ORIENTED DESIGN OF THE DYNAMIC MODEL

A. DEFINE THE PROBLEM

A system is required that will interface with existing SPY-1A Radar Controller modules and simulate the Signal Processor of the Radar. The required interface will actually include the Radar Output Module and the Radar Return Module, and the Beam Stabilization Modules. The Signal Processor Simulator must contain a database representing the environment the Radar will probe for target tracks. The database must be user changeable at any given time during the operation (i.e. add target tracks, delete target tracks, and change target tracks) so that the logical operation of the SPY-1A Radar Modules (Radar Scheduling and Track Processing) can be tested and explored.

B. DEVELOP AN INFORMAL STRATEGY

The database for the signal processor will capture the information for each target at discrete time intervals needed to define it's position. The information maintained about each target track will include it's actual position (x,y,z,r) and it's acceleration components (ax,ay,az,ar) at a discrete time interval (t). Interaction operations that a user may request include - initiation of a target track over a range of time (Ti --> Tn), deletion of a previcusly entered target track throughout all or part of it's initiated range of time, and changing a previously defined target track at any time during it's pre-defined time range. The user will also be able to start and stop the simulation at any time. The user will have a two dimensional display of

the radar environment with current tracks and relative positions symbolized during the simulation. A status report of current targets will be available while in a non-running mode to assist the user in the environment definition.

C. FORMALIZE THE STRATEGY

1. Identify the Objects and their Attributes

a. SIMULATION_OPERATIONS

b. TRACK_DATA:

- Target Information:
  - target_ID
  - actual_position
  - acceleration
  - time

2. Identify Operations on the Objects

a. SIMULATION_OPERATIONS

b. DISPLAY:

- Start
- Stop

c. TRACK_DATA:

- Status_Report
- Quit
- Target_Information:
  - create
  - delete
  - change
- Database

3. Establish the Interfaces

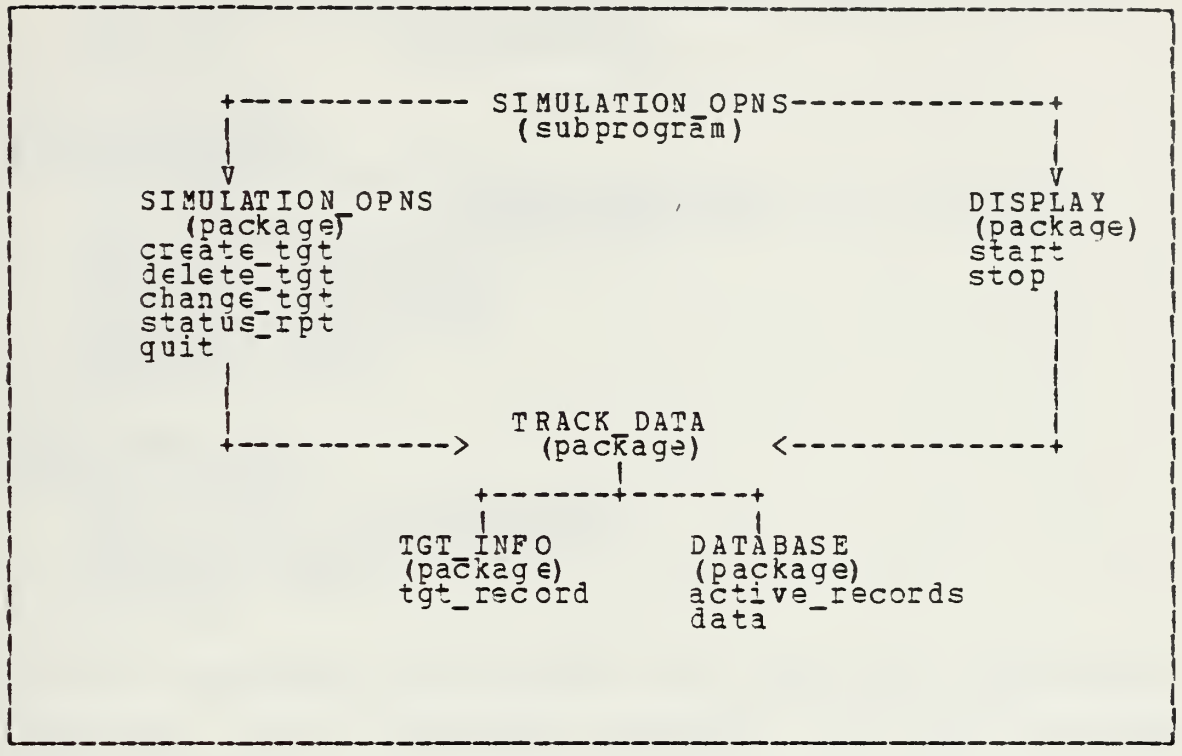

Figure E.1 Object-Oriented System Graph.

4. Code the Package Specifications in Ada

```
package TRACK_DATA is
    package TGT_INFO is
    end TGT_INFO;
    package DATABASE is
    end DATABASE;
end TRACK_DATA;

package TGT_INFO is
    type END_TIME is constant := 1000;
    type COORDINATES is (X,Y,Z,R);
    type ACCEL_VECTORS is (AX,AY,AZ,AR);
    type DISCRETE_TIME is range .. END_TIME;
    type TARGET is record
        LOCATION     :  COORDINATES;
        ACCELERATION:  ACCEL_VECTORS;
        TIME         :  DISCRETE_TIME;
        end record;
end TGT_INFO;
```

```
package DATABASE is
    use TGT_INFO;
    MAX_RECORDS  :  constant := 20;
    type RECORD_INDEX is range 0 .. MAX_RECORDS;
    type TRACK_RECORDS is array (RECORD_INDEX) of TARGET;
    ACTIVE_TGTS  :  RECORD_INDEX;
    DATA         :  TRACK_RECORDS;
end DATABASE;

with TRACK_DATA;
use  TRACK_DATA;
package SIMULATION_OPNS is
    type OPERATION is (CREATE_TGT,DELETE_TGT,CHANGE_TGT,
                       STATUS_RPT,QUIT);
    function GET return OPERATION;
    procedure CREATE_TGT;
    procedure DELETE_TGT;
    procedure CHANGE_TGT;
    procedure STATUS_RPT;
    procedure QUIT;
end SIMULATION_OPNS;

with TRACK_DATA;
use  TRACK_DATA;
package DISPLAY is
    type CONTROL is (START,STOP);
    function RUN return CONTROL;
    procedure START;
    procedure STOP;
end DISPLAY;
```

Further programming would design and build the subprograms, functions, and procedures defined by these package specifications.

APPENDIX F

SIGNAL PROCESSOR MODEL USERS MANUAL (VER.1.0)

A. GENERAL

This manual is for use with the NPS AEGIS Modeling Group AN/SPY-1A Radar Controller Model: Signal Processor Emulation version 1.0. It does not explain the structure of the modules that make up the program, only it's functional components and how they might be utilized to test the SPY-1A Model. For further information about the program design and implementation, see Kersh,T.B., Signal Processor Interface Simulation of the AN/SPY-1A Radar Controller, Masters Thesis, Naval Postgraduate School, Monterey California 1983.

The manual is divided into the two major functional areas: developing the target database to be stored in the REMEX Data Warehouse, and running the Static Model of the Signal Processor against a simple SPY-1A simulator. It will be assumed that any potential user of this system is familiar with the boot procedure for the Remex Data Warehouse disk system. Assuming the user has booted from the REMEX B: drive and logged into the REMEX D: drive, place the Signal Processor system disk in the C: drive, and type:

```
D>C:RSP <return>
```

At this point the Signal Processor Emulation System will load and the remaining database development and model operation will be menu driven.

The following functions are available within the Signal Processor Interface Simulation -

TARGET DATABASE:

1. CREATE the inital target-list and initial database.

2. DELETE any targets at any specified descrete time.
3. CHANGE the parameters of the parametric equations representing the target tracks at any specified descrete time.
4. PRINT the current target-list tc the terminal screen cr the printer at the specific descrete time represented by the target-list.

SIMULATICN:

1. RUN will execute the Static Model in a test environment to be used for testing the Signal Processor Interface Simulation System.

After development, the user can document the targets contained in the target-list at a particular descrete time , so that he has a hard-copy record of the trend of his database. This feature will be important in determinaticn of the effect of different target combinations and densities on the SPY-1A Controller Model. The Signal Processor Interface Simulaticn Target-Database development system should be usable in ccnjuction with other testing systems devised by future AEGIS Grcup members for the logical testing of the SPY-1A system.

B. CCNSTRUCT TARGET DATABASE

1. Main Menu

Just pricr tc the display of the main menu, the program will ask for user input defining the descrete time intervals to be used for the update of the buffer used by the Target-Database system. The ratio of dwell commands received from the Radar Schedular Module to the target-buffer update, multiplied by the actual turnaround time of the SP-1A Ccntroller Model will be the real-time achieved by the system. The user may assign values from .1 to 1 to this ratio value. The next question asked of the user is how

long the simulation will run. The maximum possible length is dependent on the storage space available on the REMEX Data Warehouse, and the time is based on the average assumed dwell command interval time received from the SPY-1A Radar Controller Model. To determine the simulation run limit in terms of descrete time increments, one must realize that each descrete time increment is in one-to-one correspondance with the sectors used to record the database on the REMEX. Therefore, since there are 39 sectors per track, and 210 tracks available for use, there are 8190 available descrete time intervals available for a simulation run. The real-time length for the simulation run is then dependent on the

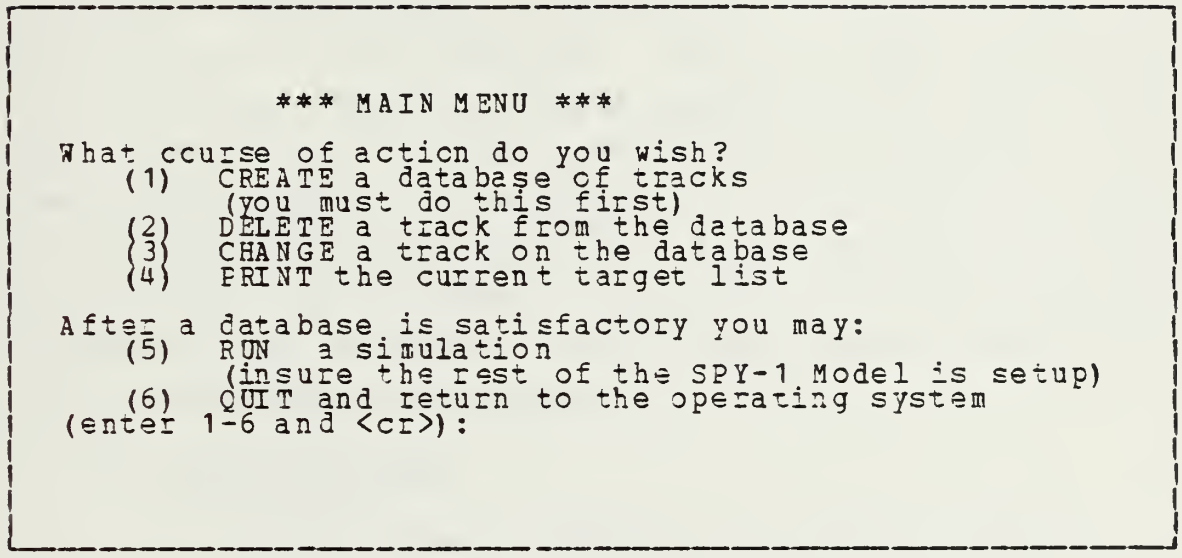

Figure F.1 Signal Processor Emulation Main Menu.

descrete time ratio. Assuming a negligible time for updating the target buffer from the REMEX, and a turnaround response time from the SPY-1A Controller Model of .001 seconds, if a ratio of "1" were chosen, the maximum time available for a simulation run would be:

1. "1" second = .001 sec. * 1000 (or 1000 dwell commands issued per buffer update)
2. since the target-buffer is updated once every second, there are 8190 seconds of maximum simulation time available.

The next item appearing on the screen is the Main Menu. The first thing required is to build a database in the REMEX Data Warehouse. To do this, the user will initially pick choice (1) CREATE. After initializing his Database, the user be able to move forward in descrete time and delete target tracks completely, or just change the parameters of the track. It is suggested that the user use option (4) PRINT after each iteration of the previous two options and after CREATE, to maintain a record of the modifications made on the database. When the user has finished with his Target Database, he may request to (5) RUN a Static Model simulation. In this mode the SPY-1A Controller Model Simulator "SPYTEST" is designed to test the Static Model. Further instructions on the use of this option are discussed in Section C. Of course, at any time after the user has returned to the Main Menu, he may choose option (6) QUIT to return to the operating system.

2. Create Database

To use the Signal Processor Interface Simulator, a Target-Database must first be constructed. A Target-List is used which contains target data to construct a Target-Database. The parameters used to set up the Target-List for each target are the constant values used in the parametric equations shown in Figure F.3 These parametric equations derive from Boone, N.A., A Multimicroprocessor Approach to simulate I/O for the AEGIS AN/SPY-1A Radar Controller, Masters Thesis, Naval Postgraduate School, Monterey California, 1981. Boone's work concerns the

```
=== CREATE TARGET MODULE ===

Initiate target #
   Parametric Equations? (1,2,3,or 4): ___
   X_range (a)? (-256,+256)nm: ___
   Y_range (u)? (-256,+256)nm: ___
   X_velocity (b)? (-32,+32)m/sec: ___
   Y_velocity (v)? (-32,+32)m/sec: ___
   X_acceleration (c)? (-.015625,+.015625)m/sec/sec: ___
   Y_acceleration (w)? (-.015625,+.015625)m/sec/sec: ___
   Z_altitude (d)? (0,20000)ft: ___

create more targets?(Y or N): ___
```

Figure F.2 CREATE Function Menu.

```
(1)  x(t) = a + b*t + c*t*t
     y(t) = u + v*t + w*t*t
     z(t) = d

(2)  x(t) = a + b*t + c*t*t
     y(t) = u + v*sin(w*t)
     z(t) = d

(3)  x(t) = a + b*cos(c*t)
     y(t) = u + v*t + w*t*t
     z(t) = d

(4)  x(t) = a + b*cos(c*t)
     y(t) = u + v*sin(w*t)
     z(t) = d
```

Figure F.3 Parametric Equations.

simulation of the AEGIS Command and Decision functions, and these equations were utilized to maintain compatiblity throughout the Model. After defining the parametric equation for the first target, the user may choose to define

further targets and will be prompted similarly as previously shown. When he is satisfied that the target-list is complete, he may indicate that no more targets are to be created, and he will be returned to the Main Menu. At that time it is recommended that the user request a PRINT of the initial target-list for future reference.

3. Delete Targets

```
=== DELETE TARGETS MODULE ===

    WHAT TARGET DO YOU WISH TO DELETE? '
      (TGT. NUM. RANGE 1-___): ____
```

Figure F.4 DELETE Function Menu.

Prior to the Delete Menu, the user will be asked "At what time do you want to delete a target?". The user is being asked to define the descrete time within his previously defined range of descrete time that he wishes to delete a previously defined target. It is important that the user have developed a plan for target modifications based on his defined descrete time range, since the Target-Database development routine will not allow one to recover deleted targets. After answering the time question, the Delete menu will be displayed, and the target list appropriately updated. After Deleting a target, the user will be prompted to "continue (Y/N)?". He may answer Y(es) to delete more targets or N(o) to return to the Main Menu.

4. Change Targets

The Change choice from the Main Menu will first prompt the user requesting what time he wants to change a target, within his predefined descrete time range. After answering, the user will see the Change menu (see Figure

```
=== CHANGE TARGETS MODULE ===

   WHAT IS THE TARGET NUMBER YOU WISH TO CHANGE?
     (TGT.NUM. RANGE 1-____): ____

   WHAT DATA ITEM IS TO BE CHANGED?
      (1) PARAMETRIC EQUATION
      (2) EQUATION PARAMETERS ___

(if the choice is one, then:)
   WHAT IS THE NEW EQUATION NUMBER (1-4)? ___

(if the choice is two then:)
   WHAT ARE THE NEW PARAMETERS:
      X_range (a)? (-256,+256)nm: ___

            . . .

      Z_alt. (d)? (0,20000)ft: ___

(and for either choice..)
DO YOU WISH TO CHANGE ANOTHER TARGET?
  (Y/N): ___
```

Figure F.5 CHANGE Function Menu.

F.5). Again, let it be emphasised that it is important to have a plan for the overall target database since it is not possible to gracefuly go backward in sequential time as a target database is developed. Also, it is again recommended to the user to obtain a print of the Target-List as soon as you return to the Main Menu.

C. RUN STATIC MODEL

The SPY-1A Controller Model Simulator "SPYTEST.CMD" is provided as a tool to test the Radar Signal Processor Interface Simulation Static Model. "SPYTEST.CMD" is just a simple eventcount and sequencer module. It contains a delay loop to simulate the time between the receipt of a "raw data" message from the Signal Processor Interface, the subsequent processing of the target data, and the resultant dwell command message generated to the Signal Processor

```
    === RSP STATIC MODEL ===
      version 1.0 June 83

At this point you should have created
a database and are now ready to run
the Static Model.

    === STATIC MODEL MENU ===
(1) TEST run the simulation
(2) QUIT and return to main menu
enter 1-2 and <cr>: ___
```

Figure F.6 STATIC MODEL Function Menu.

Interface. The delay loop is arbitrarily configured at this time, and the user should consider contriving a delay that more closely represents the turnaround time the SPY-1A system should provide. When entering the test mode, the user will be prompted to "Load SPYTEST.CMD from another system CRT/SEC. When complete, enter "0"<cr> to begin ".

When the SPY-1A Controller Simulator has been initiated, the Static Model will begin operation after the user has typed "0<cr>". The display for the Static Model is shown in

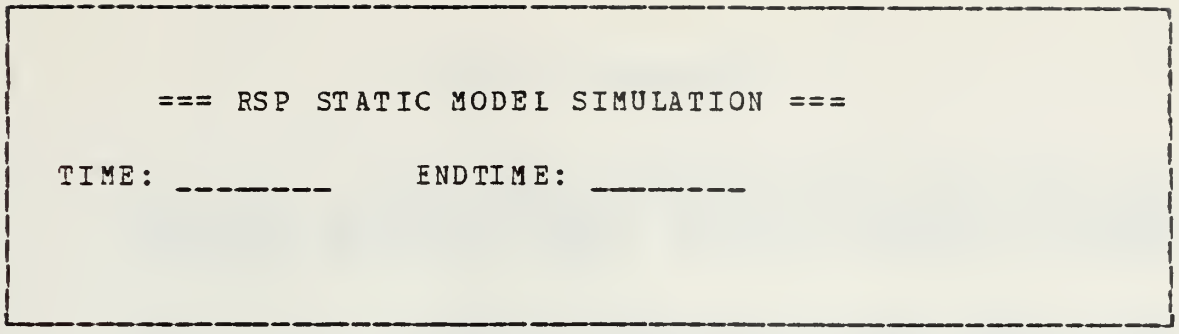

Figure F.7 STATIC MODEL Display.

Fig. F.7, and provides the user with only a minimum ammount of information to determine the progress and speed of execution for the SPY-1A Model. Since the Static Model and its inherent display functions will be part of the timed data gathered by the user, it is recommended that the SPY-1A Controller Simulator be utilized to measure the Static Model time. The measured Static Model run time can be used in future run-time testing of the NPS SPY-1A Controller Model to determine net SPY-1A Controller Model achievable speed.

LIST OF REFERENCES

1. Grant, J.V. III, A Multi-Microprocessor Based Model of the Aegis AN/SPY-1A Radar Control: Radar Scheduler Process, Master's Thesis, Naval Postgraduate School, Monterey California, 1981.

2. Cech, J.V., A Multi-Microprocessor Based Model of the Aegis AN/SPY-1A Radar Control: Track Processing, Master's Thesis, Naval Postgraduate School, Monterey California, 1982.

3. Boone, N.A., A Multimicroprocessor Approach to Simulate I/O for the AEGIS AN/SPY-1A Radar Controller, Master's Thesis, Naval Postgraduate School, Monterey California, 1981.

4. Booch, G., Software Engineering with Ada, Benjamin/Cummings Inc., 1983.

5. Riche, R.S. and C.E. Williams, A Software Foundation for AN/SPY-1A Radar Control, Master's Thesis, Naval Postgraduate School, Monterey California, 1981.

6. Almquist, T.V. and D.S. Stevens, Alteration and Implementation of the CP/M-86 Operating System for a Multi-user Environment, Master's Thesis, Naval Postgraduate School, Monterey California, 1982.

7. EX-CELL-O Corporation, REMEX Technical Manual for Data Warehouse Models RDW 3100, RDW 3200, 1979.

8. EX-CELL-O Corporation, REMEX Product Reference Manual and Performance Specifications, 1979.

9. Klinefelter, S.G., Implimentation of a Real-Time, Distributed Operating System for a Multiple Computer System, Master's Thesis, Naval Postgraduate School, Monterey California, 1982.

10. Micropolis Corporation, Micropolis Specification - 1220 Series Rigid Disk Drive Subsystems, Chatsworth, California, 1980.

11. Digital Research Corporation, CP/M-86 Operating Systems Guide, 1981.

12. Digital Research Corporation, PL/I-86 Manual, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Defense Logistic Studies Information Exchange U.S. Army Logistics Management Center Fort Lee, Virginia 23801	1
3.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5.	Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
6.	Captain Brad Mercer, USAF, Code 52Zi Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	CPT Todd B. Kersh HQ, U.S. Army CECOM ATTN: DRSEL-TCS-CR Fort Monmouth, New Jersey 07703	2
8.	RCA AEGIS Data Repository RCA Corporation Government Systems Division Mail Stop 127-327 Moorestown, New Jersey 08057	1
9.	Library (Code E33-05) Naval Surface Warfare Center Dahlgren, Virginia 22449	1
10.	Daniel Green (Code N20E) Naval Surface Warfare Center Dahlgren, Virginia 22449	1
11.	Curricular Officer, Code 37 Computer Technology Curricular Office Naval Postgraduate School Monterey, California 93940	1
12.	Dr. M.J. Gralia Applied Physics Laboratory Johns Hopkins Road Laurel, Maryland 20707	1
13.	Dana Small Code 8242, NOSC San Diego, California 92152	1

14. CPT Mark R. Kindl, U.S.A. 1
413 E. Washington St.
Villa Park, Illinois 60181

15. Dr. Bert Y. Kersh 1
260 Sacre Lane
Mcnmcuth, Oregon 97361

201684
Thesis
K3894
c.1
Kersh
Signal processor interface simulation of the AN/SPY-1A radar controller.